Ingeniería Técnica de Telecomunicación:
Sonido e Imagen

Universidad
Carlos III de Madrid
www.uc3m.es

Proyecto fin de carrera

Aplicación de telefonía corporativa de la UC3M

Autora: Estefanía Sarasola Elvira
Tutor: Israel Gutiérrez Rojas

Universidad Carlos III de Madrid
Octubre 2013

Agradecimientos

A mis padres y mi hermano en primer lugar, que siempre me han apoyado en todo aquello en lo que he decidido embarcarme, y al resto de mi familia, tanto los que ya no están como los que siguen aquí. Y a mi gata Albir, que en tantas largas noches de trabajo me ha hecho compañía (aunque no vaya a poder leer este agradecimiento nunca, jeje).

A todos mis amigos, y gente maravillosa que he conocido a lo largo de estos años, tanto dentro como fuera de la universidad. Cada uno habéis aportado un poquito a mi vida cuanto menos, y muchos habéis dejado una huella grande en mí que me ha llevado a ser en gran parte la persona que soy hoy en día.

En especial quería agradecer a mis amigos más cercanos, aquellos que habéis seguido ahí tanto en los buenos momentos como en los malos y que me habéis tenido que aguantar en mis días más pesados y en mis días con exceso de energía y felicidad (que para el caso estoy igual de pesada, jajaja). Sois bastantes, pero ya deberíais saber quiénes sois, así que simplemente diré que especialmente a Pedro de Oro Martín, Ana de Prado Navarrete y Gladys Carrillo, por todo vuestro apoyo y cariño durante todos estos años.

A mi tutor Israel, por sus valiosos consejos y enseñanzas y su ayuda en este último tramo de mi carrera.

A Asunción Merino por darme la oportunidad de desarrollar la aplicación de este proyecto mientras trabajaba en la universidad a su cargo.

A Son Goku, por mantener la Tierra a salvo mientras redactaba esta interminable memoria y permitirme que pueda acabar de una vez la carrera. Gracias Goku, ¡qué haríamos sin ti!
Y a Sara Villanueva que es la causante de este párrafo, además de mi primera mentora en Android, SVN y muchas otras cosas durante mi primer trabajo.

Por último, dar las gracias a la comunidad de stackoverflow que desinteresadamente tanto nos ayuda a los que nos estamos iniciando o llevamos pocos años aún en el desarrollo de software, fuente de sabiduría donde siempre encuentras si no la solución exacta siempre pistas para solucionar el problema que tengamos.

A todos, ¡gracias!

Resumen

Este proyecto fin de carrera trata del desarrollo de una aplicación para el sistema operativo de dispositivo móviles Android encargada de poner a disposición del personal que trabaja en la Universidad Carlos III de Madrid (PAS/PDI) parte de los servicios más destacados del área de telefonía de dicha universidad.

Estos servicios se pueden desglosar en tres grandes grupos: consulta de la facturación y tarifas de las líneas fijas y móviles asociadas al personal de la universidad, consulta de la configuración de los smartphones y tablets homologadas por la universidad y acceso al servicio de soporte de la universidad, y por último consulta de la configuración y activación del servicio de telefonía IP de la universidad.

Dado su carácter de aplicación que va a ser usada en un ámbito real, este proyecto abarca desde sus inicios realizando las primeras versiones del diseño en mockups hasta pruebas sistemáticas del código implementado, pasando por correcciones de diseño y funcionalidad tras estudiar la usabilidad de ciertos elementos o cambios en los requisitos de los clientes, entre otras necesidades de este tipo de proyecto. Además, ha sido necesaria la comunicación con distintos colaboradores externos, afectando a distintas áreas del desarrollo del proyecto.

Abstract

This final degree project consist on the development of an application for the Android operating system that provides part of the services of the telephony department of the Carlos III University to the university‘s employees (PAS/PDI).

These services covers three groups of functionality: a) inquire landline and mobile telephone billing and fares of the employees of the university; b) inquire smartphone and tablet approved by the university configuration info and access to the university support service; and c) inquire the VoIP service activation and configuration info.

As this project is a real application that is going to be used in a real scope, it covers from its origins with the early versions of the mockups design to the systematic testing of the code, passing through design and functionality improvements after studying the usability of certain elements or changes in the customer requirements, among other requirements of this kind of project. Furthermore, it has been needed to communicate with external collaborators, impacting in different areas of the project development.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Durante el tiempo que la autora de este proyecto fin de carrera estuvo trabajando en el área de telefonía de la Universidad Carlos III de Madrid (UC3M) se decidió facilitar al personal de la universidad que disponen de una línea telefónica, fija o móvil, asociada a la universidad (PAS/PDI - personal de administración y servicios y personal de investigación-, empresas externas, institutos, másteres y doctorados) el acceso a sus datos de facturación telefónica corporativa, para que tuvieran un mayor control del dinero gastado mes a mes, ya que en definitiva, afecta al presupuesto del departamento.

Para ello, la universidad ha contratado a una empresa externa que está realizando una página web en la que consultar los datos de la facturación de la universidad emitida por la operadora de telefonía con la que tiene contratado el servicio de telefonía la universidad. Sin embargo, desde el departamento de telefonía se le quería dar un impulso mayor a este servicio, por lo que se decidió dotarlo de movilidad desarrollando una aplicación para dispositivos móviles, la cual es el objeto de estudio de este proyecto fin de carrera.

Por otro lado, aunque la función más importante y común de la aplicación será la consulta del consumo de telefonía, también se decidió incluir más información útil del servicio de telefonía de la universidad como es el servicio de telefonía IP y las guías de configuración de los smartphones y tablets homologadas por la universidad.

1.2. Objetivos

El objetivo de este proyecto es desarrollar una aplicación para el sistema operativo Android que ofrezca los servicios más útiles del área de telefonía de la UC3M para el personal de la universidad, de forma sencilla de usar tanto para usuarios ya acostumbrados al uso de smartphones como para usuarios que llevan poco tiempo utilizando uno, como es el caso de algunos usuarios de la universidad.
Los servicios ofrecidos en la aplicación por el servicio de telefonía se engloban en tres grandes grupos:

- Consulta de información sobre su facturación telefónica y de las tarifas de la universidad.

- Guías de configuración de los smartphones y tablets homologados por la universidad y acceso al soporte de la universidad.

- Activación y guía de configuración del servicio de telefonía IP de la universidad.

Alguno de los objetivos que me han motivado a realizar este proyecto han sido los siguientes:

- Ganar soltura en la realización de aplicaciones Android partiendo de los requisitos o mockups del cliente o colaborador.

- Desarrollar una aplicación para un uso final real, por lo que ha de ser estable. Además su código ha de ser fácilmente mantenible y adaptable a posibles cambios a lo largo del tiempo y por otros desarrolladores, no el que creó la aplicación en primer lugar.

- Como en toda aplicación real, se ha de aprender a trabajar con distintos colaboradores como en este caso ha sido la diseñadora encargada de crear los iconos de los menús y definir algunos de los colores utilizados en la aplicación. También la empresa externa que realizará los servicios web de consumo y tarifas. Además se trata de un trabajo para un cliente final, rol que en este caso han ejercido los distintos responsables del servicio de telefonía y los asesores de contenidos de la UC3M. Como clientes del proyecto, han sido los encargados de la validación de la aplicación y de su interfaz de usuario.

- Desarrollar la aplicación utilizando las herramientas que ofrecen las nuevas versiones de Android para el desarrollo dinámico en una misma aplicación para smartphones y tablets o smartphones de gran tamaño.

- Aprender a utilizar las librerías de compatibilidad para poder implementar las opciones que permiten las nuevas versiones de Android en dispositivos móviles con versiones más antiguas.

- Desarrollar una interfaz fluida, que mantenga informado al usuario en todo momento de las operaciones que se están realizando.

- No sobrecargar de forma excesiva cada pantalla con mucha información que agobie al usuario. Para ello se ha realizado un buen diseño de las pantallas con mucha cantidad de información como las de la normativa, el consumo de grupo o el asesor de tarifas.

- Desarrollar una aplicación robusta y escalable, ya que el número de usuarios es alto y variable.

- Utilizar web services (servicios web) para la obtención de la mayor cantidad de información posible en la aplicación. Por ejemplo el nombre de las tarifas o el texto de los avisos de telefonía que incluye el menú principal de la aplicación, para que el usuario no tenga que actualizar la aplicación cada vez que hay un cambio en el nombre de las tarifas, o de los avisos entre otros posibles cambios.

- Aprender a utilizar las librerías de test de Android y Robotium para la creación de pruebas sistemáticas.

En la presente memoria se recoge el procedimiento realizado y las tecnologías utilizadas para la consecución satisfactoria de la aplicación de telefonía corporativa de la Universidad Carlos III de Madrid (UC3M), cumpliendo estos objetivos y otros que han aparecido durante el desarrollo de este proyecto.

1.3. Contenido

La presente memoria se estructura en una serie de seis capítulos distribuidos de la siguiente manera:

- En el capítulo 1 se ha hecho una introducción del proyecto, hablando sobre la motivación y los objetivos que se querían alcanzar durante el desarrollo del mismo.

- En el capítulo 2 se hace un repaso del estado del arte de todas las tecnologías utilizadas durante el desarrollo de este proyecto. Sobre todo se ha ahondado en las características de Android como sistema operativo y los elementos de sus librerías más utilizados en la aplicación de telefonía corporativa de la UC3M. Además se habla sobre el problema de la fragmentación en Android y sus consecuencias a la hora de desarrollar una aplicación en este sistema operativo. También se hace mención a los web service, tecnología utilizada en este proyecto para la consulta y obtención de información. Por último, se hace un repaso a algunas consideraciones importantes cuando se desarrolla para un dispositivo móvil, sea para el sistema operativo móvil que sea.

- En el capítulo 3 se explican las distintas partes que han conformado el proyecto en sí y de qué modo se han utilizado cada una de las tecnologías y librerías presentadas en el estado del arte del capítulo anterior. Se explica en detalle cada una de las funcionalidades de la aplicación desarrollada en el proyecto. Por último, se explican los mayores problemas a afrontar durante el desarrollo del proyecto debido a la necesidad de crear una aplicación multidispositivo.

- En el capítulo 4 se exponen las pruebas realizadas durante el desarrollo del proyecto. Estas pruebas se dividen en dos tipos principalmente, pruebas manuales realizadas en el dispositivo móvil directamente, y pruebas sistemáticas que se ejecutan desde el IDE Eclipse.
- En el capítulo 5 se realiza un repaso a la historia del proyecto. Se habla del tiempo dedicado a cada fase del proyecto y del presupuesto estimado del mismo.

- En el capítulo 6 se muestran las conclusiones tras realizar el proyecto, y las líneas futuras previsibles para mejorar la aplicación de telefonía corporativa de la UC3M.

Además se incluye un glosario de términos con el significado de todas las siglas utilizadas en la presente memoria, así como la bibliografía utilizada durante su elaboración.

Capítulo 2

Estado del arte

2.1. Introducción

Los teléfonos inteligentes, o más comúnmente llamados *smartphones*, están en el mayor momento de crecimiento de su historia. Se conoce como smartphone a teléfonos móviles capaces de conectarse a internet, gestionar el correo, manejar contenidos multimedia, en definitiva, almacenar información y manejarla como un pequeño ordenador de bolsillo.

Estos teléfonos inteligentes han ido aumentando en calidad y potencia de hardware llegando a superar, e incluso empezando a sustituir en el caso de las tablets a los PCs en algunos hogares en los que sólo utilizaban éste para navegar por Internet, entrar en las redes sociales y poco más.

Se puede observar este gran crecimiento en los últimos análisis presentados por la compañía analista de mercado IDC sobre el primer trimestre de 2013, en los que se ha declarado que por primera vez se han vendido más dispositivos smartphones que dispositivos sin estas capacidades.
En concreto, IDC estima que unos 480 millones de teléfonos móviles fueron puestos en circulación en los tres primeros meses del año, de los cuales el 51,6% corresponden con smartphones [1].

Gran parte del éxito de los dispositivos inteligentes es su gran variedad, tanto en dimensiones (smartphone, tablet, phablet) cada uno de ellos con multitud de tamaños, como de precio, la elección entre distintos sistemas operativos, la gran personalización en definitiva que hace que cualquier persona pueda elegir el dispositivo más adecuado a sus necesidades.

En cuanto a sistemas operativos móviles hay diversas opciones, aunque destacan como más utilizados sobre todo Android y iOS. Otros como Blackberry o Windows Phone siguen manteniéndose. Además, hay nuevos emergentes muy prometedores como Firefox OS y Ubuntu mobile o Sailfish entre otros. Sin embargo, el más consolidado a nivel mundial es Android, además de ser el que mayor crecimiento está teniendo como puede verse en los resultados obtenidos en el segundo cuatrimestre del año 2013 respecto a los del 2012 según los analistas de IDC en la Tabla 1:

Top Smartphone Operating Systems, Shipments, and Market Share, Q2 2013 (Units in Millions)

Operating System	2Q13 Unit Shipments	2Q13 Market Share	2Q12 Unit Shipments	2Q12 Market Share	Year-over-Year Change
Android	187.4	79.3%	108	69.1%	73.5%
iOS	31.2	13.2%	26	16.6%	20.0%
Windows Phone	8.7	3.7%	4.9	3.1%	77.6%
BlackBerry OS	6.8	2.9%	7.7	4.9%	-11.7%
Linux	1.8	0.8%	2.8	1.8%	-35.7%
Symbian	0.5	0.2%	6.5	4.2%	-92.3%
Others	N/A	0.0%	0.3	0.2%	-100.0%
Total	**236.4**	**100.0%**	**156.2**	**100.0%**	**51.3%**

Tabla 1 - Tabla de ventas de smartphones del segundo cuatrimestre de los años 2012 y 2013 y el porcentaje de mercado que abarca los cinco principales sistemas operativos de móviles (extraído de [2])

Queda clara la gran importancia que está cobrando la movilidad hoy en día, y parece que seguirá aumentando a lo largo de los años, de ahí la necesidad de dar a los usuarios una forma de ver la información que desean de forma adecuada para el dispositivo que estén utilizando.

Dado que el sistema operativo de smartphones más extendido entre los teléfonos corporativos de la universidad es Android, se decidió crear en primer lugar la aplicación de telefonía corporativa de la UC3M para dicho sistema operativo. Con esta aplicación, los miembros de la UC3M podrán acceder a información relevante del servicio de telefonía de la universidad de una forma cómoda, rápida y en cualquier lugar.

2.2. Android

2.2.1. Introducción

Android es un sistema operativo libre y de código abierto (bajo licencia Apache) que trabaja principalmente con un kernel de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores; y la máquina virtual Dalvik, similar a la de Java pero optimizada para que pueda correr eficientemente en dispositivos móviles (pocos recursos) y de la que se pueden crear varias instancias de forma eficaz.

Los usuarios interactúan con el dispositivo móvil a través de una pantalla, táctil normalmente, y haciendo uso de aplicaciones. Algunas de estas aplicaciones ya vienen preinstaladas en el sistema operativo, otras las añaden los fabricantes con sus capas de personalización, y otras las añaden las operadoras de telefonía. La mayoría de las aplicaciones las instala el usuario descargándolas de los markets de aplicaciones, principalmente el de Google, el Play Store, pero también de otros como el de Amazon. Por último, el usuario puede decidir descargar el paquete de la aplicación desde páginas webs de terceros u otros medios e instalarlo directamente. Esta última práctica no es muy recomendable por la alta posibilidad de instalar malware.

2.2.2. Inicios de Android

Android fue desarrollado inicialmente por la compañía Android Inc., la cual fue comprada por Google en 2005.

En su inicio Android estaba pensado para ser un sistema operativo para cámaras fotográficas digitales inteligentes, pero vieron que este nicho de mercado no era lo suficientemente grande, por lo que decidieron desarrollar un sistema operativo para dispositivos móviles que rivalizara con los sistemas que en aquel momento estaban en auge, Symbian y Windows Mobile (iOS aún no se había lanzado). [3]

Durante los siguientes años los rumores sobre que Google quería entrar en el mercado de la telefonía móvil fueron aumentando, hasta su anuncio definitivo el 5 de noviembre de 2007 por parte de la Open Handset Alliance, un consorcio de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles anunció entre las que se incluyen Google, HTC, Samsung, T-Mobile, Qualcomm o Nvidia entre otras.

2.2.3. Versiones

Android, como cualquier otro sistema operativo saca actualizaciones a menudo. En este caso suele haber varias al año, una grande que suele significar el paso a una nueva versión con un nuevo nombre comercial (ej. Gingerbread o Ice Cream Sandwich), y actualizaciones de mejoras menores que sólo llevan asociadas un cambios en el número de versión pero no de nombre comercial. Cada uno de los nombres comerciales hace referencia a un tipo de dulce distinto: Donut (v. 1.6), Gingerbread (v. 2.3.x), Ice Cream Sandwich (v. 4.0.x), Jelly Bean (v. 4.1.x y 4.2.x), etc.

Figura 1 - Dibujo de la evolución de las versiones de Android (extraído de [4])

Sin embargo existe un gran problema en el caso de los dispositivos móviles, y que en el caso de Android se agrava al tener tanta cantidad de fabricantes distintos. Esto junto a las distintas especificaciones de hardware tanto en tamaños, como distintos tipos de sensores de las cámaras, o distintos sensores incorporados constituyen la denominada *fragmentación* en Android.

Además, ya no sólo se reducen a dispositivos móviles (smartphones o tablets) sino que Android ya está presente en otro tipo de dispositivos como son cámaras digitales (como la Galaxy Camera de Samsung), relojes inteligentes (como el SmartWatch de Sony) o gafas de realidad aumentada (como las Google Glass de Google), por lo que tiene que ser un sistema capaz de lidiar con tantas diferencias.

Volviendo al problema de las actualizaciones de Android para los dispositivos móviles, el principal problema es que los fabricantes deben adaptar la nueva versión del sistema operativo al hardware específico de cada terminal. Además, prácticamente todos los fabricantes de dispositivos móviles en Android incluyen su propia capa de personalización del sistema operativo para intentar dar un valor añadido a sus dispositivos móviles, con servicios y aplicaciones extras. Algunos ejemplos son la capa Touchwiz de Samsung o la Sense de Htc.
Esto implica que aunque Google saque una nueva actualización de su sistema operativo, estas actualizaciones pueden llegar con muchos meses o años de atraso, incluso en muchos dispositivos, aunque por potencia de hardware podrían tener la nueva versión, nunca la tendrán porque no le ha parecido rentable al fabricante invertir tiempo en ello.
Además, si el terminal no es libre, la operadora también mete su parte de personalización a la versión para meter sus propios servicios y aplicaciones, por lo que se retrasa aún

más la actualización e incluyen aplicaciones que en muchos casos el usuario final no está interesado en ellas ni las va a usar, y ocupan un tamaño no aprovechable ya que además no suelen poderse desinstalar sin modificar la versión de fábrica del sistema operativo.

Esto ha provocado que aunque Android vaya mejorando, y esté ya en la versión 4.2 Jelly Bean, la realidad es que casi la mitad de los dispositivos Android del mundo están aún en versiones anteriores como se muestra en la Tabla 2 y en la Figura 2 a día 2 de Octubre del 2013.

Version	Codename	API	Distribution
2.2	Froyo	8	2.2%
2.3.3 - 2.3.7	Gingerbread	10	28.5%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	20.6%
4.1.x	Jelly Bean	16	36.5%
4.2.x		17	10.6%
4.3		18	1.5%

Tabla 2 - Distribución de versiones de Android (extraído de [5])

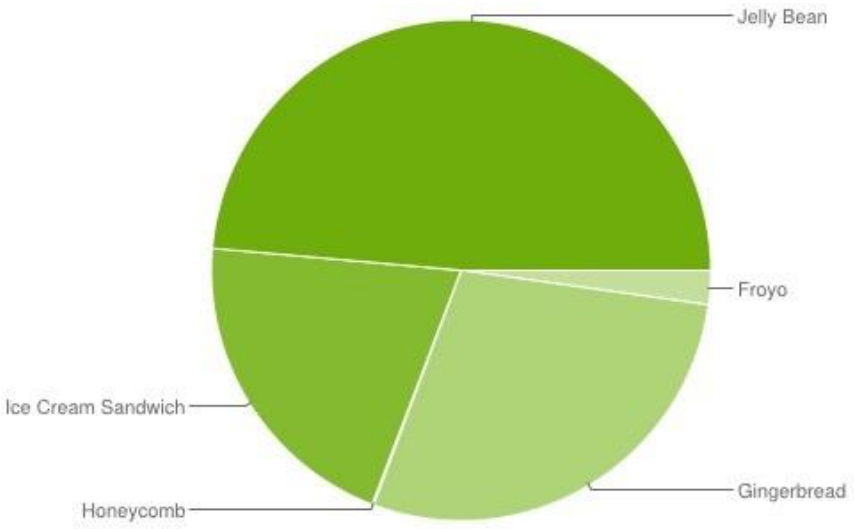

Figura 2- Gráfico de distribución de versiones de Android (extraído de [5])

Esto hace tener que tomar una decisión en cuanto a la plataforma para la que desarrollar y la relación ventaja/problemas que acarrea cada una de las decisiones.

2.2.4. Arquitectura

Los componentes que forman Android se agrupan en capas, utilizando cada una de estas capas elementos de la capa inferior para realizar sus funciones. Por ese motivo, a este tipo de arquitectura se le denomina pila.

En el Figura 3 se puede observar la pila software de Android:

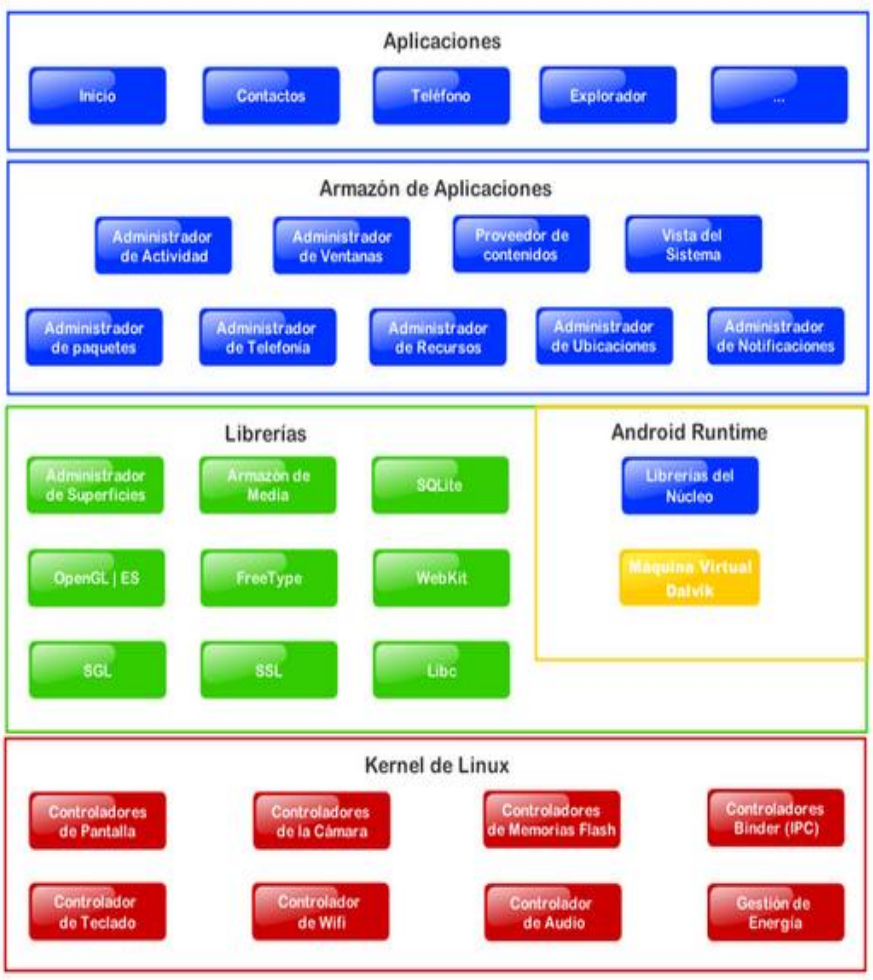

Figura 3 - Pila software de Android (extraído de [6])

A continuación se describen las características de cada capa:

- **Kernel de Linux**

El núcleo del sistema operativo Android es una adaptación de un kernel Linux versión 2.6. A partir de la versión 4.0 Ice Cream Sandwich de Android hasta la última 4.2.X Jelly Bean por ahora se utiliza como base un kernel Linux 3.0.

El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura, lo que permite que se pueda acceder a esos componentes, como por ejemplo la cámara o la brújula, sin necesidad de conocer el modelo o características precisas de los que están instalados en cada teléfono. Para cada componente del hardware, el fabricante debe crear los controladores o *drivers*, librerías específicas de ese componente que se incluyen en el kernel para que pueda ser manejado desde el software de Android.
Además, el kernel se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (networking), etc.

- **Librerías**

La siguiente capa que se sitúa justo sobre el kernel la componen las bibliotecas nativas de Android. Estas librerías están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono, tarea que normalmente realiza el fabricante, que también se encarga de instalarlas en el terminal antes de ponerlo a la venta. Su cometido es proporcionar funcionalidad a las aplicaciones, para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma más eficiente en principio.
Muchas de las librerías, esenciales para el funcionamiento de Android, utilizan proyectos de código abierto. Sin embargo, en muchos casos éstas son propietarias al igual que los *drivers* de cada fabricante, lo que ha llevado, sumado a otras razones como que Google Play (la tienda de aplicaciones de Google) acepte aplicaciones que no sean gratuitas, a una escisión de Android llamada Replicant [7] que utiliza sólo librerías de código abierto y su propia tienda de aplicaciones sólo gratuitas para conseguir un sistema operativo completamente de código libre y gratuito.

A continuación se habla brevemente de algunas de las librerías más importantes:

- Surface Manager: Se encarga de componer las imágenes que se muestran en la pantalla a partir de capas gráficas 2D y 3D.

- OpenGL ES: Se encarga de los gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil aceleración hardware, o motor software optimizado si no dispone el terminal de la aceleración. Es una versión ligera del OpenGl utilizado para PCs.

- SGL: Proporciona gráficos en 2D, por lo que será la librería más utilizada habitualmente por la mayoría de las aplicaciones.

- Librerías multimedia: Proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas,

etc.). Permiten visualizar, reproducir e incluso grabar en los formatos de imagen, vídeo y audio más comunes como JPG, GIF, PNG, MPEG4, AVC (H.264), MP3, AAC o AMR.

- FreeType: Permite mostrar fuentes tipográficas, tanto basadas en mapas de bits como vectoriales.

- Librería SSL: Proporciona servicios de encriptación Secure Socket Layer en la comunicación segura a través de Internet.

- Librería SQLite: Permite la creación y gestión de bases de datos relacionales. Versión ligera para dispositivos móviles de bases de datos SQL [8].

- Librería WebKit: Proporciona un motor para las aplicaciones de tipo navegador como el actual navegador incluido por defecto en la plataforma Android.

- Librería libc: Se trata de una derivación de la librería BSD (Berkeley Software Distribution) de C estándar (libc), adaptada para dispositivos embebidos basados en Linux. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

- **Entorno de ejecución (Android Runtime)**

El entorno de ejecución de Android, aunque se apoya en las librerías anteriormente comentadas, no se considera una capa en sí mismo, dado que también está formado en parte por bibliotecas. Estas librerías contienen la mayor parte de las bibliotecas Java más utilizadas, además de otras propias de Android.

Por otro lado, el entorno de ejecución incluye la máquina virtual Dalvik, basada en la máquina virtual de Java, pero más ligera y optimizada para recursos limitados como los de un dispositivo móvil. Las aplicaciones se compilan en el formato .dex (Dalvik executable), formato comprimido hasta un 50% más que el formato .class que ejecuta la máquina virtual de Java convencional y que permite además de ahorrar espacio en el dispositivo móvil, una carga más rápida. Como se corre en una máquina virtual, el código sólo se compila una vez, y podrá ejecutarse en cualquier dispositivo móvil que tenga una versión de Android mayor o igual para la que fue desarrollada, y en ciertos casos en los que sea compatible también en versiones menores.

Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik como medida de seguridad para aislar unas aplicaciones de otras, método denominado *sandbox*, del que se hablará en la sección 2.2.6. Delega al kernel de Linux algunas funciones como threading (manejo de hilos) y el manejo de la memoria a bajo nivel.

- **Framework de Aplicaciones**

Esta capa la forman todas las clases y servicios que las aplicaciones utilizan directamente para realizar sus funciones. Todas las aplicaciones para Android, tanto las nativas del sistema operativo, como las propias del dispositivo, o las desarrolladas por cualquier usuario, utilizan estas bibliotecas. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik. Entre las más importantes se encuentran las siguientes:

- Administrador de actividades (Activity Manager): Controla el ciclo de vida de las *actividades* y la pila de *actividades*. Ambos conceptos se explican en las secciones 2.3.6 y 2.3.7. respectivamente.

- Administrador de ventanas (Windows Manager): Organiza lo que se muestra en pantalla del dispositivo.

- Proveedor de contenidos (Content Provider): Permite que una aplicación pueda compartir su información con otras aplicaciones. Se explica en la sección 2.3.1.

- Vistas (Views): Son los elementos que proporciona Android para crear la GUI (Graphical User Interface o interfaz de usuario), tales como botones, listas, o mapas de Google Maps.

- Administrador de paquetes (Package Manager): Las aplicaciones Android se distribuyen en paquetes (archivos .apk). Esta biblioteca permite obtener información sobre los paquetes actualmente instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes.

- Administrador de telefonía (Telephony Manager): Permite acceder al hardware relacionado con la telefonía, para el envío de SMS/MMS y realizar llamadas telefónicas.

- Administrador de recursos (Resource Manager): Permite gestionar los recursos de las aplicaciones, tales como imágenes, cadenas de texto, o los archivos con la disposición de las vistas dentro de una actividad (layout). Además se encarga de elegir los recursos adecuados si hay diversos recursos alternativos en función por ejemplo del tamaño de la pantalla o el idioma del dispositivo.

- Administrador de ubicaciones (Location Manager): Proporciona a las aplicaciones información de localización y posicionamiento haciendo uso del GPS y/o de las redes Wi-Fi y/o de telefonía disponibles.

- Administrador de notificaciones (Notification Manager): Permite que las aplicaciones puedan avisar al usuario haciendo uso de notificaciones como alertas

en la barra de estado, hacer vibrar el dispositivo móvil, o activar los LEDs del dispositivo móvil si dispone de ellos.

- **Aplicaciones**

Esta capa contiene todas las aplicaciones de las que dispone el dispositivo móvil, tanto las nativas del sistema operativo, como las instaladas por los fabricantes y operadores de telefonía, como las instaladas por el usuario.

2.2.5. Lenguajes de programación

Las aplicaciones Android se escriben principalmente del lenguaje de programación Java y se utiliza el lenguaje XML para definir los recursos de la aplicación.

El NDK de Android permite escribir parte del código en C o C++, sin embargo aumenta en gran medida la complejidad del código, y no aporta grandes beneficios, por lo que su uso está indicado para casos en los que sea esencial, y nunca porque el desarrollador prefiera programar en dicho lenguaje.

2.2.6. Seguridad en Android

La seguridad en Android se asienta sobre varios pilares. El primero de ellos, del cual ya se habló anteriormente en la sección 2.2.4 de Arquitectura de Android, se basa en que cada aplicación se ejecuta en su propio proceso de Linux. Por otro lado, cada proceso se ejecuta en su propia máquina virtual Dalvik, por lo que corre aislado de otras aplicaciones, lo que en informática se conoce como *sandbox*. Por lo tanto, cada aplicación tiene por defecto acceso sólo a los componentes que necesita para realizar su trabajo, pero no a aquellas zonas a las que no tiene permiso como el código o datos privados de otras aplicaciones, creando un ambiente de ejecución (runtime) seguro. Ya que esta medida de seguridad es a nivel del kernel del sistema operativo, el código nativo está también incluido en el sandbox de aplicación, a diferencia de los sandbox típicos de linux.
Cada aplicación instalada en el sistema es un *usuario* diferente, con su propio *user ID* (UID, identificador de usuario) por defecto, el cual no es conocido por la aplicación si no sólo por el sistema. A todos los archivos y al código de una aplicación se les establece permisos de acceso sólo por el ID asignado para dicha aplicación por el sistema, para que sólo ésta pueda acceder a ellos. Si una aplicación intenta acceder o realizar algo para lo que no tiene permisos el sistema se protegerá contra ello y finalizará la ejecución de la aplicación al momento.

Por otro lado, si hay algún error de corrupción de memoria, éste sólo afectará en el contexto de la aplicación en el que se ha generado dicha corrupción sin comprometer al resto del sistema como sucede en otros sistemas operativos.

Para el paso de archivos entre aplicaciones, Android hace uso de diversas herramientas, como la posibilidad de que dos aplicaciones compartan el mismo ID, pudiendo ser ejecutadas a su vez en el mismo proceso Linux y en la misma máquina virtual. Sin embargo, ambas aplicaciones deben estar firmadas con el mismo certificado, por lo que sólo se puede utilizar este método con otra aplicación del mismo desarrollador o empresa, y no con aplicaciones de terceros, ya que podría resultar en un grave problema de seguridad en ese caso. Otro método de permitir compartir datos entre aplicaciones es el uso de *content providers* que se verán en la siguiente sección. Por último, nuevamente a través de permisos para acceder a archivos del dispositivo tales como la cámara, los mensajes SMS o la tarjeta de memoria externa SD (Secure Digital).

Esto lleva al segundo pilar, y es que Android basa su seguridad en un sistema de permisos. Algunos ejemplos de permisos son el uso de la cámara, el acceso a Internet, el acceso a los contactos del teléfono o hacer uso del GPS (Global Positioning System o sistema de posicionamiento global).

Cada aplicación debe registrar los permisos que necesita para funcionar en su archivo `AndroidManifest.xml`, del cual se hablará en la siguiente sección sobre conceptos básicos de Android. En el proceso de instalación de una aplicación Android se avisa al usuario del dispositivo móvil sobre cuáles son los permisos que dicha aplicación necesita para funcionar, y es decisión del usuario final el decidir instalarla o no. Una vez instalada no se volverá a avisar al usuario de los permisos que se le han concedido a dicha aplicación, aunque puede comprobarlos en la configuración de las aplicaciones del dispositivo. Si la aplicación es desinstalada, al instalarla de nuevo volverá a aparecer la pantalla de aceptación de los permisos.

Si una aplicación intenta acceder a una funcionalidad para la cual no tiene el permiso necesario declarado en su archivo `AndroidManifest`, se generará una excepción de seguridad y el sistema cerrará la aplicación.

2.3. Conceptos básicos de Android para el desarrollo de la aplicación de telefonía corporativa de la UC3M

En esta sección se van a definir los principales elementos que forman parte de Android y algunos menos comunes pero que han sido utilizados en el desarrollo de la aplicación de telefonía corporativa de la UC3M. Obviamente Android tiene muchos más elementos, por lo que sólo se hablará sólo de aquellos utilizados en el desarrollo de la aplicación que

estudia esta memoria. Sin embargo, se hablará brevemente de algunos elementos importantes aunque no se hayan hecho uso de ellos. Con esta sección se pretende dar al lector una base para entender las explicaciones más técnicas del capítulo 3, en la que se habla del uso de estos componentes en la aplicación.

2.3.1. Componentes de aplicación

Los componentes de aplicación conforman los bloques fundamentales en una aplicación Android. Cada uno de estos componentes facilita un punto de entrada para el sistema operativo a la aplicación desarrollada, pero no siempre para el usuario de dicha aplicación. Además, en muchos casos dependen unos componentes de otros para el correcto funcionamiento de la aplicación.

Hay cuatro tipos de componentes de aplicación, cada uno encargado de un propósito distinto, y con su propio ciclo de vida que define cómo el componente es creado y destruido.

- **Activity**

Una *activity* o actividad representa una pantalla de la aplicación, que lleva asociada una interfaz gráfica de usuario con la que los usuarios pueden interactuar. Son las distintas pantallas que se muestran al usuario durante su uso de la aplicación. Una aplicación está compuesta por una o más actividades.

Un ejemplo de una actividad sería por ejemplo la lista de canciones a reproducir en el reproductor de música, o la pantalla para escribir un email.

Una actividad se implementa como una subclase de `Activity`. De esta clase se hablará más en profundidad más adelante.

- **Service**

Un servicio es un componente que se ejecuta en segundo plano, y que no tiene interfaz gráfica de usuario. Se utiliza para llevar a cabo procesos de larga duración o para realizar trabajo para procesos remotos. Otro componente, como por ejemplo una actividad, puede inicializar un servicio de dos maneras: uno simple o bien uno con el que el componente puede interactuar.

Un ejemplo de servicio sería por ejemplo el encargado de reproducir la música en el dispositivo mientras que el usuario puede estar en otra aplicación distinta a la del reproductor.

Un servicio se implementa como una subclase de `Service`.

- **Content provider**

Un proveedor de contenidos o content provider maneja datos de una aplicación que están compartidos para que otras aplicaciones puedan acceder a ellos. Estos datos pueden estar almacenados en cualquier almacenamiento persistente que una aplicación pueda acceder como por ejemplo el sistema de ficheros del dispositivo móvil, una base de datos SQLite, o desde una web. Estos datos a los que se les permite el acceso al content provider podrán ser leídos e incluso modificados por otras aplicaciones si el content provider tiene los permisos requeridos para ello.

Un ejemplo de un proveedor de contenidos es el que se encarga de manejar la información de los datos de contacto del usuario del dispositivo móvil. La aplicación que quiera hacer uso de este content provider tendrá que declarar que desea tener estos permisos y el usuario del dispositivo móvil ha de aceptarlos a la hora de instalar la aplicación como ya se vio en la sección 2.2.6. Seguridad en Android.

Aunque su mayor uso está en la compartición de datos entre aplicaciones, también puede ser útil para leer y escribir datos de forma privada para una única aplicación.

Las aplicaciones que quieren acceder a un content provider lo han de hacer a través de un *content resolver*, una clase capaz de manejar los contenidos del dispositivo móvil.

Un proveedor de contenido se implementa como una subclase de `ContentProvider` y debe implementar una serie de métodos que permitan a otras aplicaciones realizar transacciones sobre los datos de dicho proveedor de contenidos.

- **Broadcast receivers**

Es un componente que responde a los mensajes de difusión, normalmente producidos por el sistema como por ejemplo que queda poca batería o que una imagen ha sido capturada por la cámara. Sin embargo, también pueden ser enviados por aplicaciones, por ejemplo para permitir que otras aplicaciones sepan que cierto tipo de datos se ha descargado y pueden usarlos.

Aunque no tengan interfaz gráfica de usuario pueden crear notificaciones en la barra de estado del dispositivo. Normalmente se utilizan para inicializar otros componentes, como por ejemplo un servicio, cuando se recibe un evento en concreto.

Un broadcast receiver se implementa como una subclase de `BroadcastReceiver`. Para enviar un broadcast receiver se envía como un objeto `Intent`, del cual se habla en la sección 2.3.4.

2.3.2. AndroidManifest

Para que el sistema pueda iniciar algún componente de una aplicación (`Activity`, `Service`, `ContentProvider` y `BroadcastReceiver`), se debe haber declarado ese componente en el archivo de manifiesto de la aplicación (`AndroidManifest.xml`).
El sistema lee dicho manifiesto del cual obtiene, de entre otras cosas, la siguiente información que ha debido ser declarada:

- Los componentes de los que hace uso la aplicación para que puedan ser inicializados.

- Los permisos de seguridad que la aplicación necesita, como acceder a internet o el poder hacer llamadas telefónicas.

- El nivel mínimo de la versión de Android que necesita un dispositivo móvil Android para poder ejecutar la aplicación.

- El hardware y características usadas o requeridas por la aplicación como la cámara o el bluetooth. Se diferencia entre usado y requerido en que para funcionar correctamente la aplicación hace uso de esa característica del teléfono, como por ejemplo el giroscopio, pero si ponemos el uso del giroscopio como característica usada, seguirá siendo visible dicha aplicación en el market para todos los dispositivos móviles aunque no tengan un giroscopio. Sin embargo, si se pone como característica requerida, para todo dispositivo móvil que no posea un giroscopio la aplicación se mostrará como que no compatible y no dejará instalarla. En ocasiones es conveniente no ponerlo como requerida, ya que siguiendo el ejemplo anterior, puede que la funcionalidad con el giroscopio sea una pequeña parte de la aplicación y el resto de la aplicación pueda usarla un dispositivo sin giroscopio perfectamente.

- Las librerías que usa la aplicación aparte de las propias del sistema Android, tales como la librería de Google Maps.

2.3.3. Recursos de la aplicación

Las aplicaciones Android hacen uso de más componentes aparte del código, como por ejemplo de imágenes, archivos de audio, o cualquier cosa relacionada con el aspecto visual de la aplicación como los colores, las animaciones o los layouts (ficheros de diseño de los que se hablará más adelante). Cuando se compila la aplicación, estos recursos se compilan eficientemente y se empaquetan con la aplicación.

En Android se proporciona el mecanismo de los recursos externos que permite separar en gran medida la interfaz gráfica del usuario con respecto a la lógica del código y los datos.

Tener separados los recursos de aplicación hace más sencillo el poder cambiarlos por otros sin tener que modificar el código. Además gracias al uso de los recursos alternativos se facilita el poder desarrollar la aplicación para distintos dispositivos según el tamaño de la pantalla, o la densidad de píxeles por pantalla, pero también según el estado en el que se encuentre el dispositivo, como por ejemplo el tener un distinto diseño del interfaz de usuario según esté en vertical (portrait) o en horizontal (landscape) o que según el idioma del dispositivo se utilice el fichero XML con los textos en un idioma u otro.

Para cada recurso que se incluye en el proyecto de Android, el SDK build tools genera un identificador único (ID) en el proyecto para que pueda ser referenciado desde otros recursos o desde el código y se almacena en la clase `R`. Esta clase `R` no puede ser cambiada manualmente. Si se cambia algo de ella, dará un error y se regenerará. Por ejemplo para una imagen `icon.png` guardada en el directorio `res/drawable/` se generaría un ID llamado `R.drawable.icon` para ser llamado desde código, o se utilizaría `@drawable/icon` si se quisiera referenciar desde otro recurso en XML.

Las librerías de cada una de las versiones de Android contienen sus propios recursos, algunos de ellos públicos que cualquiera puede utilizar en sus propias aplicaciones. Por ejemplo, el icono de buscar nativo de Android se referencia desde código con `android.R.drawable.ic_menu_search` y desde otro recurso en XML como `@android:drawable/ic_menu_search`.

La potencia que tiene el uso de estos recursos alternativos es que siempre se utiliza el ID del recurso, tanto en el código como en los archivos XML. Este ID está asociado al nombre de dicho recurso, y este nombre es el mismo en las distintas alternativas de un mismo recurso. Es el sistema el que se encarga de escoger el más acertado según las características del dispositivo que se está usando.

Siguiendo el ejemplo de la imagen `icon.png`, se dijo antes que estaría en el directorio `res/drawable/`, pero si se quiere añadir distintas imágenes para que se utilice la más adecuada y que por ejemplo en pantallas de alta densidad no se pixele en exceso la imagen al hacer el escalado hasta la densidad de pantalla que usa el dispositivo, se habría de tener imágenes de distintas resoluciones en los directorios `res/drawable-`

`ldpi/`, `res/drawable-mdpi/`, `res/drawable-hdpi/` y `res/drawable-xhdpi/`. A estos sufijos se les denomina *qualifier*. El nombre para cada una de las imágenes es el mismo, por lo que el identificador será el mismo y el sistema será el encargado de elegir el que mejor se adapte basándose en los cualificadores (*qualifier*) de cada recurso alternativo si es que se provee alguno.

Todos los recursos se han de disponer en el subdirectorio adecuado de la carpeta `res/`. Un ejemplo simple de esta estructura que siguen los recursos sería la siguiente:

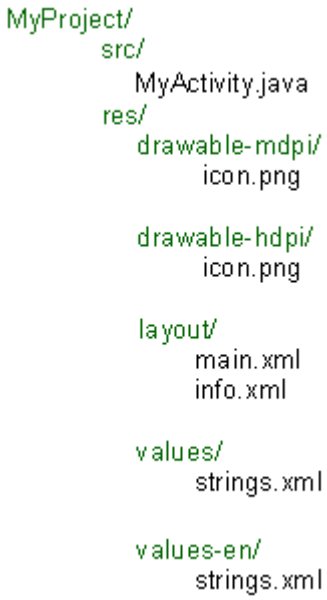

Figura 4 - Estructura parcial de un proyecto Android para mostrar el uso de carpetas de recursos alternativos

Se puede consultar la tabla 2 en la página web de Android Developer sobre “Providing Resources” [9] para ver todos los posibles cualificadores que puede tener una carpeta de recursos alternativos. Se pueden añadir varios cualificadores a una misma carpeta para recursos alternativos más concretos, pero ha de ser en el orden de aparición de dicha tabla, pues es el orden en que el sistema comprueba los cualificadores.

Las carpetas de recursos que se soportan actualmente son: animator, anim, color, drawable, layout, menu, raw, values y xml. A continuación se va a explicar en más detalle aquellos recursos que han sido utilizados en el desarrollo de la aplicación de estudio de esta memoria.

- **anim**

Archivos XML para crear animaciones. Existen dos tipos de animaciones:

  - Tween animation: Se crea una animación aplicando una serie de transformaciones a una imagen (como por ejemplo de translación, rotación, cambio en el tamaño, etc.). Se utiliza para ello la clase `Animation`.

  - Frame animation**:** Se crea una animación mostrando una serie de imágenes una tras otra. Se utiliza para ello la clase `AnimationDrawable`.

- **color**

Archivos XML que definen listas de colores en función del estado, o *selector*. Estas listas de colores en función del estado son archivos XML que permiten cambiar el color del objeto a cuyo fondo (*background*) se le aplique este selector en función de su estado. Por ejemplo, un botón pasa por varios estados, el de por defecto o inactivo o presionado. O un campo para la entrada de texto (*`EditText`*) puede tomar el estado de por defecto o inactivo, el de que tiene el foco y está siendo presionado, o el de que tiene el foco pero no está siendo presionado. Se le asigna un color distinto a cada estado, y en función del estado se aplicará un color u otro al fondo del botón o del campo de texto editable en los ejemplos anteriores. Dichos colores pueden estar definidos en hexadecimal con el formato que se explica en la sección 2.3.3 “Recursos: values” en el apartado de color, o bien hacer una referencia a un recurso de color que se haya definido en otro fichero XML de color a través de @color/nombre_del_color_ya_definido.

- **drawable**

Los drawables son los gráficos de una aplicación en Android. Pueden ser archivos de imagen tales como .png, .jpg, .9.png o .gif, o bien gráficos creados en archivos XML como los *shape* o los *state list.*

Los archivos de imagen de mapa de bits (.png, .jpg y .gif) crean *`BitmapDrawable`*.

El formato .9.png crea *`NinePatchDrawable`*. Utiliza imágenes .png con zonas que se pueden expandir dependiendo de su contenido cuando se han de reescalar a un tamaño mayor.

Los *shape* son archivos XML para crear formas geométricas básicas como rectángulos con o sin borde, colores o gradientes.

Los *state list* son selectores de imágenes en función del estado del elemento gráfico (botón, campo de texto, etc.) similar a los selectores de color explicados en la sección anterior 2.3.4.2.

Hay más subtipos de *drawables* y modos de crear éstos, pero los anteriormente nombrados son los usados principalmente en el desarrollo de la aplicación de telefonía corporativa de la UC3M.

- **layout**

Los layouts son archivos XML que definen la interfaz de usuario de ciertos elementos. Pueden ser la interfaz de usuario completa de una actividad o bien sólo la interfaz de un botón por ejemplo, el cual se utiliza en distintas partes de la interfaz de usuario o en incluso otras actividades.

Un ejemplo del primer caso sería por ejemplo la definición completa de la interfaz gráfica de usuario de una pantalla de inicio de sesión, con su campo de texto editable (`EditText`) para introducir el usuario, otro campo de texto editable para la contraseña y un botón (`Button`) con el texto "Enviar". Cada componente gráfico mencionado en el ejemplo hereda de la clase View, de la cual se hablará en profundidad en la sección 2.3.10. Todos estos campos se definirían en un fichero XML que se asociará por código a la actividad encargada del inicio de sesión.

Un ejemplo del segundo caso sería por ejemplo la definición gráfica de una fila de una lista personalizada. Las interfaces gráficas para listas de las que se disponen como base en Android sólo contienen texto, por lo que si por ejemplo queremos mostrar una lista de opciones, pero cada una ha de llevar una imagen asociada a la izquierda y el color ha de ser azul claro, se ha de crear un layout que sea un campo de texto (`TextView`) con el color de fondo. La imagen en función de si es la misma para todas las opciones o no, se incluirá en dicho layout o tendrá que añadirse por código. Por último se asocia por código este layout en lugar del básico de Android.

En la mayoría de los casos se utilizará una subclase de `ViewGroup`, un contenedor para componentes gráficos básicos denominados View, de los que se hablará más en detalle en la sección 2.3.10. Los tipos de objetos `ViewGroup` más utilizados son `LinearLayout`, `RelativeLayout` y `FrameLayout`.

En algunos casos en los que se va a incluir un layout dentro de otro que ya tiene un padre o raíz adecuada, como por ejemplo al hacer uso de `<include>` dentro de un fichero layout en los recursos (no por código) se puede utilizar `<merge>` para obtener un elemento raíz alternativo que no será dibujado. De este modo se ahorra el tener Views que no se necesitan realmente en la arquitectura final de la interfaz de usuario y que ralentizan la carga de dicho layout.

Los atributos de altura (`android:layout_height`) y de anchura (`android:layout_width`) son siempre obligatorios de definir para que el sistema sepa cómo ha de dibujarlos en la pantalla. Habría un error en tiempo de ejecución en caso de no establecer un valor válido.

- **values**

Los recursos values son archivos XML que contienen valores tales como cadenas de caracteres, colores, o dimensiones.

En estos recursos values hay incluidos diversos tipos de recursos. Se pueden poner en el mismo fichero XML o bien, por claridad, se suelen poner en distintos ficheros en función de su tipo. Existen nombre convencionales en función del tipo:

- arrays.xml: Valores de arrays tipados. Se pueden tener distintos tipos de arrays como de drawables o de colores. Para los más utilizados como son las cadenas de caracteres y de enteros hay tipos ya definidos: `string-array` e `integer-array`.

- colors.xml: Valores de colores. Los colores se definen en formato hexadecimal: #AARRGGBB para colores que contengan información sobre el canal alpha para medir la cantidad de transparencia del color, o en el formato #RRGGBB en caso de que no se desee incluir información de transparencia, por lo que será un color sólido. También se puede dar en el formato #ARGB y #RGB. La R hace referencia al componente rojo del color, G al verde y B al azul.
  En internet se pueden encontrar multitud de información y de aplicaciones para obtener el color que se desea en hexadecimal [10].

- dimens.xml: Valores de dimensiones, como por ejemplo 16px (16 píxeles), o 5dp (5 píxeles independiente de la densidad o *density-independent pixel*). En la sección 2.4.3 se habla más en profundidad del tipo de dimensiones en Android.

- strings.xml: Fichero en el que se incluyen las cadenas de caracteres que se utilizan en toda la aplicación. Se puede encontrar más información de cómo utilizar este recurso alternativo para dotar a la aplicación de distintos idiomas en la sección “2.2.4. Diferentes idiomas”.

- styles.xml: Fichero en el que se puede definir una serie de atributos para los componentes gráficos de la interfaz y que crea estilos reutilizables en distintos componentes de la interfaz de usuario (`Views`) o sobre una actividad o aplicación entera, estos dos últimos casos definiendo este estilo como un tema para dicha actividad o aplicación en el fichero `AndroidManifest`. Se utiliza sobre todo si distintos componentes gráficos tienen los mismos atributos (fondo, tamaño del

texto, color del texto, etc), se les aplica el mismo estilo y sólo ha de escribirse una vez. También es más fácil cambiar sólo en el estilo los atributos que han de modificarse posteriormente que cambiarlos uno por uno en los distintos componentes gráficos.
Los estilos pueden heredar de otros estilos con los que compartan la mayor parte de sus atributos excepto alguno que se define o redefine en el estilo hijo.

- **Otros recursos**

Existen otros tipos de recursos que se pueden externalizar en archivos XML como por ejemplo los IDs. Este tipo de recurso, dado un nombre para el ID, hace que se genere en el archivo .R un identificador único.
Este identificador se le puede dar luego a un componente gráfico, como por ejemplo a un botón que haya sido creado por código y que por lo tanto si se necesita que tenga un ID asociado se le tiene que dar por código uno que se haya creado en el fichero de IDs de los recursos. Es similar a cuando se le asigna un ID a través del atributo "id" a un componente gráfico en un layout.
Otro ejemplo de uso es para generar IDs únicos como códigos en una actividad que ha iniciado otra actividad de la que se espera un resultado.

Otro ejemplo es el uso de archivos XML para la creación de alias. Permiten crear un alias para otro archivo de recursos, es decir, se le asocia otro nombre a un recurso. Se suele utilizar cuando se quiere utilizar el mismo archivo de recursos para distintas versiones, por ejemplo `res/layout-large` para versiones anteriores a la 3.2 y `res/layout-sw600dp` para la versión 3.2 y posteriores para pantallas de grandes dimensiones (tablets y televisiones). Sin embargo para no duplicar cada recurso en cada una de las carpetas se crea en la carpeta por defecto de ese tipo de recurso el recurso para grandes pantallas y se le asigna un nombre cualquiera. Aunque este nombre no es el mismo que el del archivo por defecto (pues están en la misma carpeta por lo que no pueden tener el mismo nombre), el sistema sabrá que tiene que mostrar el recurso para las grandes pantallas en lugar del de por defecto a través del fichero de alias que se incluye en cada una de las carpetas para pantallas de gran tamaño. Se añade una archivo de alias a cada una de las carpetas `res/values-large/` y `res/values-sw600dp/` en los que se sustituye el nombre del recurso por defecto por el creado para pantallas de grandes dimensiones. De este modo sólo se tiene una vez cada recurso y no hay duplicidades innecesarias. Se suelen nombrar a estos archivos con el nombre de refs.xml.

2.3.4. Views

Las views (vistas) son los elementos gráficos básicos de toda aplicación. De esta clase heredan los componentes como botones, campos de texto, etc, pero también los `ViewGroup` que sirven de contenedores para estas views o para otros `ViewGroup` tales como `LinearLayout` o `RelativeLayout`.

La interfaz de usuario se compone por lo tanto de descendientes de la clase `ViewGroup` y descendientes de la clase View ordenados de forma jerárquica como se puede observar en el ejemplo de la Figura 5. Todos estos componentes gráficos pueden ser tanto los ya existentes en Android como los creados a partir de ellos por el desarrollador.

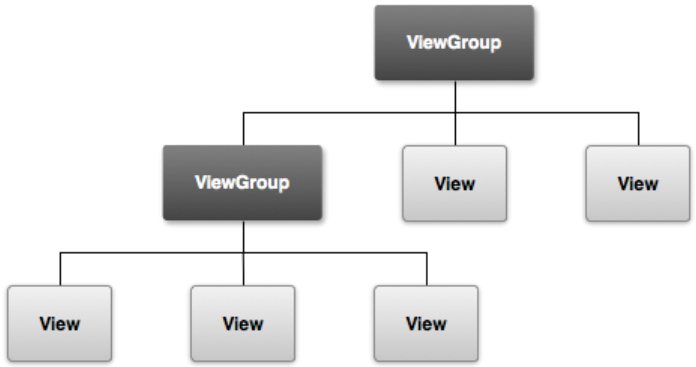

Figura 5 - Ejemplo de la jerarquía de un interfaz de usuario simple [15]

Hay multitud de componentes gráficos que se han utilizado, por lo tanto en la Figura 6 sólo se muestran algunos de los más utilizados en la aplicación:

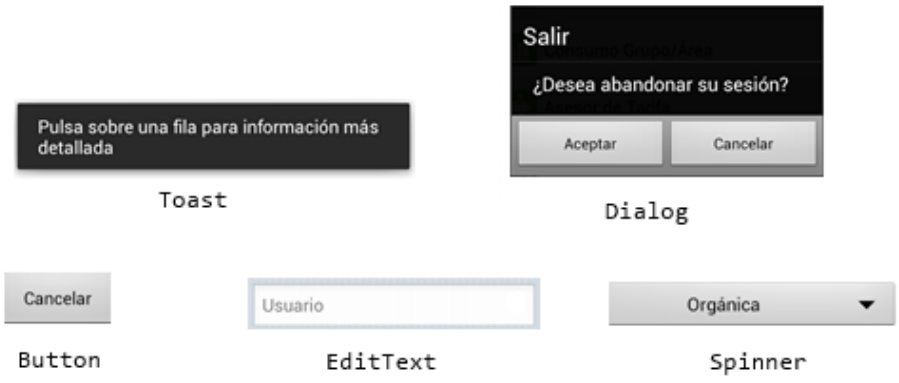

Figura 6 - Ejemplos de Views

2.3.5. Intents

De los cuatro componentes de aplicación mencionados en la sección 2.3.1, las actividades, los services y los broadcast receivers se activan con el uso de mensajes asíncronos llamados *intent*.

Un intent se crea con un objeto de la clase `Intent`, el cual puede ser explícito o implícito.
Un intent explícito especifica un componente concreto que se quiere activar como otra actividad de tu propia aplicación.
En un intent implícito se especifica un tipo de componente que se quiere activar, por ejemplo que se quiere utilizar la cámara o que se quiere abrir una página web. Si el dispositivo móvil tiene varias aplicaciones que pueden ser activadas para el uso requerido, se le muestra al usuario una lista con las opciones disponibles y es el usuario el que elige cual aplicación utilizar.

Para el caso de las actividades, se diferencian dos casos: aquel en el que la actividad que inicializa otra a través de un intent espera que ésta le devuelva un resultado y aquel en el que no espera ningún resultado. Un ejemplo del primer caso sería el usar un intent para que el usuario elija un contacto de la lista de contactos y que devuelva la URI (Uniform Resource Identifier) que apunta al usuario escogido. Un ejemplo del caso en que no se espera un resultado podría ser por ejemplo el pasar a la siguiente pantalla de una aplicación tras pulsar un botón en la pantalla.
Para el caso en el que no se espera ninguna respuesta se pasa el objeto Intent al método `startActivity()`. Si se requiere que se devuelva un resultado se ha de utilizar `startActivityForResult()` en la inicialización de la nueva actividad e implementar el método `onActivityResult()` para manejar el resultado obtenido.

2.3.6. La clase Activity

Como ya se explicó en la sección 2.3.1., una actividad es una pantalla, la cual tiene asociada una interfaz de usuario con la que éste puede interactuar.

A cada actividad se le da una ventana para dibujar la interfaz de usuario. Esta ventana se suelen ajustar al tamaño de la pantalla del dispositivo, pero también puede ser más pequeña y que se posicione encima de otras ventanas.

Una aplicación suele consistir en una serie de múltiples actividades relacionadas entre ellas. Una de ellas se define como el launcher en el archivo `AndroidManifest` del que ya se habló en la sección 2.3.2., similar al “main” típico de las aplicaciones de escritorio (*standalone*).
Es la primera pantalla que le presenta al usuario cuando inicia la aplicación si no la había iniciado antes. Una actividad puede iniciar otra y así consecutivamente, creando el flujo de

la aplicación. Cuando se inicia una actividad nueva, la anterior se para pero no se destruye, si no que se preserva en una pila de actividades llamada "*back stack*". Cuando una nueva actividad se inicia, ésta se coloca al principio de la pila de actividades, y la actividad previa pasa a la segunda posición de la pila. Cuando un usuario desea volver a la anterior actividad, presiona el botón de atrás *Back button*. La actividad actual se saca de la pila de actividades, por lo que se destruye. La pila de actividades se verá en más detalle en la sección 2.3.7.

Por lo tanto una actividad pasa por distintos estados que se le notifican a través del uso de los métodos *callback* del ciclo de vida de la actividad. Los métodos *callbacks* son métodos orientados a aplicaciones dependientes de eventos. Se registra un escuchador al cual el sistema notifica si se ha producido el evento para el cual se registró. Otro ejemplo serían los métodos `onClick()` de los botones: el sistema ejecuta el código dentro del método `onClick()` cuando el botón es presionado, es decir, sólo es ejecutado cuando se cumple el evento para el cual se registró.

2.3.7. Ciclo de vida de una actividad

Se pueden diferenciar cuatro estados en una actividad:

- Activa o ejecutándose**:** Si está en primer plano y tiene el foco de interacción con el usuario.

- Pausada: La actividad ha perdido el foco del usuario pero permanece visible, por debajo de otra ventana que no cubre toda la pantalla o es semitransparente. Una actividad pausada mantiene su estado y la información de sus componentes, pero puede ser destruida en casos de muy baja memoria del sistema.

- Parada**:** Si la actividad es ocultada por completo de la vista del usuario. Mantiene todavía todo el estado y la información de sus componentes, pero a menudo será destruida cuando el sistema esté bajo de memoria.

- Destruida**:** Cuando el sistema ha matado el proceso de una actividad que estaba pausada o parada, y se quiere iniciar de nuevo dicha actividad se ha de recrear de nuevo e intentar recuperar su estado previo.

El ciclo de vida de una aplicación lo maneja el sistema, por lo que es éste el que produce la llamada de estos métodos para informar a la actividad del estado en el que se encuentra.

El siguiente diagrama (Figura 7) muestra los distintos estados comentados. Los rectángulos grises muestran los métodos callbacks que se llaman en una actividad durante su ciclo de vida.
En estos métodos se pueden realizar las distintas operaciones como definir la interfaz de usuario de la actividad, o salvar el estado de la actividad de forma persistente si es muy importante hacerlo antes de que ésta sea destruida, o liberar recursos que estén siendo utilizados como el uso de sensores cuando el usuario no tiene en primer plano la actividad que necesita usarlos.

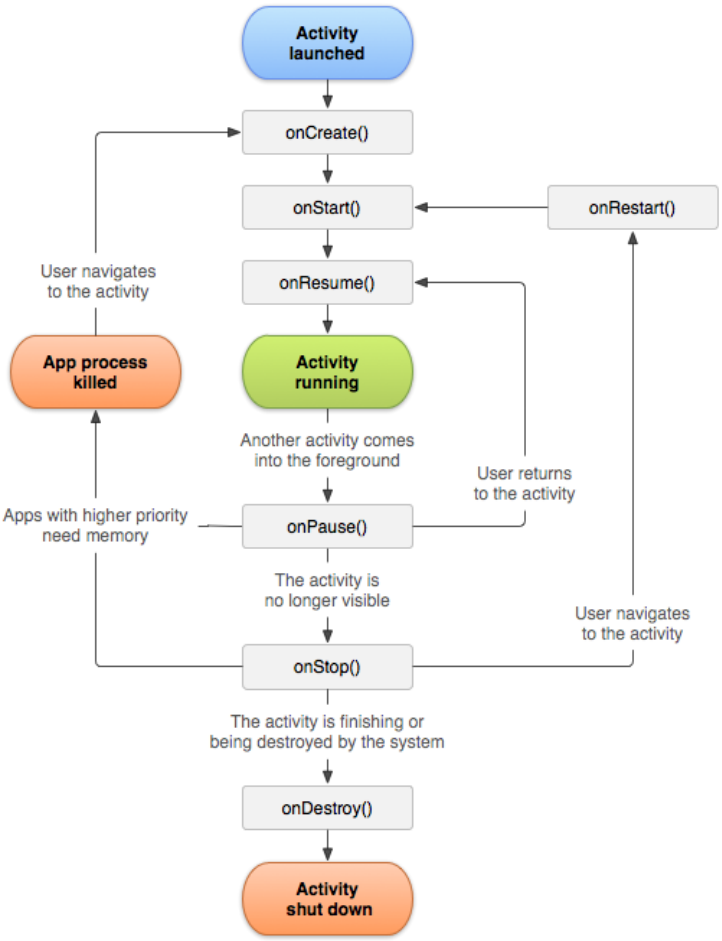

Figura 7 - Diagrama del ciclo de vida de una Activity (extraído de [11])

Como se puede observar en el diagrama, toda el ciclo completo de vida de una actividad ocurre entre la primera llamada al método `onCreate(Bundle)` hasta la llamada a `onDestroy()`.

Entre la llamada a `onStart()` y `onStop()` se abarca el ciclo de vida visible de la aplicación. En este estado la actividad es visible para el usuario pero no tiene el foco ni está en primer plano y por lo tanto el usuario no interactúa con ella. Estos métodos pueden ser llamados varias veces por el sistema según se oculte o sea visible la actividad.

El ciclo de vida en primer plano sucede entre la llamada a `onResume()` y a `onPause()`. Durante este estado, la actividad está en primer plano y el usuario está interactuando con ella. Estos métodos pueden ser llamados frecuentemente, por ejemplo cada vez que el dispositivo móvil se bloquea y se duerme por inactividad, por lo que las operaciones que se lleven a cabo en estos métodos deben ser rápidas y ligeras.

Todas las actividades deben implementar el método `onCreate(Bundle)` para hacer la configuración inicial, definiendo como mínimo cuál será su interfaz de usuario (referenciando un archivo XML creado en los recursos de layout o bien creando previamente algún elemento gráfico por código y estableciendo éste como interfaz de usuario).
El método `onPause()` también suele ser implementado en muchos casos para preparar la actividad para ser parada, almacenando la información importante de forma persistente, o liberando los recursos que no sean necesarios si el usuario no tiene dicha actividad en primer plano, como podrían ser los sensores en un videojuego.

2.3.8. Pila de actividades y tareas

Como ya se habló previamente, Android maneja el paso entre estas actividades con el concepto de *stack* o pila. Una pila es una estructura de datos cuyo modo de acceso para almacenar y recuperar datos es de tipo LIFO (*Last In First Out* o último en entrar, primero en salir). Es decir, en una pila el último elemento que entra es el primero que sale, el penúltimo en entrar es el segundo en salir y así sucesivamente.

Cuando en la aplicación se inicia una nueva actividad, por ejemplo al pulsar un botón que te lleva a una nueva pantalla, la nueva actividad se coloca al principio de la pila de actividades y la anterior actividad en la que se encontraba el usuario pasa a la segunda posición, y así sucesivamente. Esta pila de actividades se denomina "*back stack*".
Al pulsar el botón de volver atrás del teléfono (*Back button)* la actividad que estaba en primera posición, la que estaba en primer plano (mostrándose la usuario) se destruye y la anterior actividad pasa de nuevo a la primera posición de la pila y es la que pasa a estar en primer plano. Esto permite llevar un control del flujo del usuario a través de las pantallas.

En la Figura 8 se puede observar de forma gráfica el funcionamiento de la pila de actividades de Android:

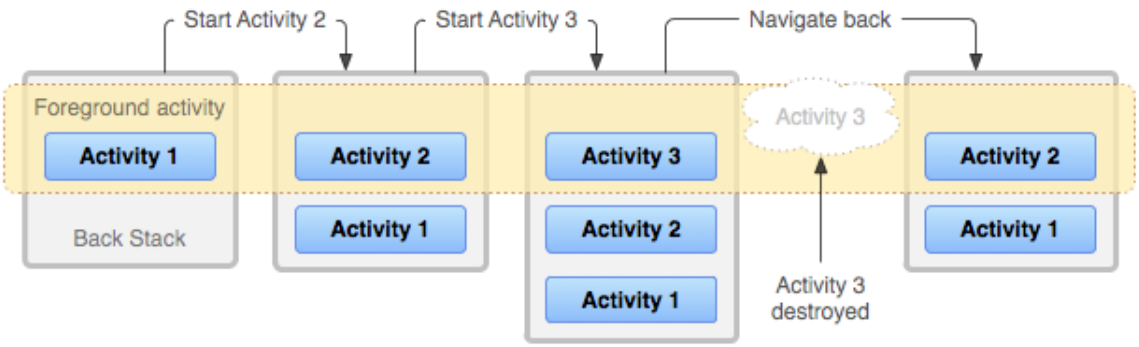

Figura 8 - Representación de cómo funciona la pila de actividades en Android. Extraído de [12]

Una tarea es una colección de actividades con las que el usuario interactúa para llevar a cabo cierto trabajo, aunque dichas actividades sean de distintas aplicaciones. Por ejemplo, cuando una aplicación quiere hacer una foto, llama a la aplicación nativa de la cámara del dispositivo y una vez tomada la foto regresa a la actividad desde la que se inició el uso de la cámara.

Una tarea, con su pila de actividades asociada, se mueve a segundo plano cuando se regresa al escritorio del dispositivo a través del botón *Home*, o bien cuando se inicia una nueva tarea (para la cual se creará su propia pila de actividades). La pila de actividades de esa tarea se mantiene intacta y conservando su estado, aunque esté en segundo plano, para que cuando el usuario decida volver a esa tarea siga tal y como estaba cuando la dejó (Figura 9). Poder tener varias tareas iniciadas a la vez e ir pasando entre ellas y que mantengan el estado en el que se dejaron permite tener lo que se conoce como multitarea en Android.

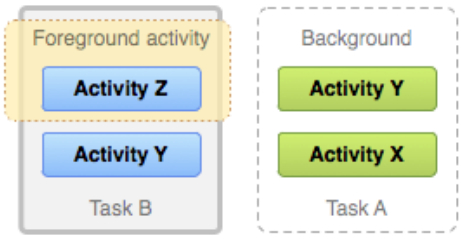

Figura 9- Representación de cómo funciona la multitarea en Android. Extraído de [12]

Sin embargo, hay que tener en cuenta que dado que se pueden tener múltiples tareas en segundo plano, el sistema puede destruir dichas tareas en caso de falta de memoria haciendo que el estado de la actividad se pierda. En la siguiente sección se hablará sobre estos casos.

2.3.9. Política de eliminación de procesos por falta de memoria y salvación del estado de una actividad

Por defecto, todos los componentes de una aplicación se ejecutan en el mismo proceso, aunque puede ser cambiado en el archivo `AndroidManifest` si hay necesidad de ello, pero no es lo más recomendado.
El sistema operativo Android sigue una política de eliminación de procesos en caso de detectar que queda poca memoria. Esta política de eliminación de procesos tiene en cuenta el estado del proceso y su interacción con el usuario para elegir qué procesos eliminar, atendiendo a los siguientes casos ordenados de menor a mayor probabilidad de eliminación:

- Procesos en primer plano: Procesos que implican o bien a una actividad con la que está interactuando el usuario en ese momento, o bien un broadcast receiver que se está ejecutando o bien un proceso que ha lanzado algún otro proceso que tiene un service que se está ejecutando aún.

- Proceso visible: Una actividad que se sigue viendo pero que no está en primer plano o no tiene el foco.

- Procesos de servicio: Un service que se está ejecutando en segundo plano.

- Procesos en segundo plano: Una actividad que no es visible para el usuario.

- Procesos vacíos: No se están ejecutando pero se mantienen en memoria para que se carguen más rápido si el usuario decide volver a ellos.

Cuando el sistema para alguna de las actividades de una aplicación (por ejemplo si se inicia una nueva actividad o si se pasa a segundo plano por volver al escritorio) el sistema puede destruir dicha actividad en casos de niveles bajos de memoria. En este caso la información sobre el estado de la actividad se pierde. Pero aunque sea destruida una actividad, el sistema sabe el lugar que ocupaba dicha actividad en la pila de actividades de su tarea, por lo que cuando se quiere mostrar de nuevo al usuario, es decir, se encuentra de nuevo en la primera posición de la pila, el sistema recreará la actividad en lugar de simplemente reanudarla. Es decir, se creará como si fuera la primera vez que se inicia, pasará por el método `onCreate()` de nuevo.

Si no se quiere perder información de lo que el usuario hubiera hecho en dicha actividad antes de pasarla a segundo plano y se destruyera, se ha de salvar el estado o bien a través del método `onSaveInstanceState()`, o bien de forma persistente en el método `onPause()`.

El método `onPause()` es el único que está asegurado que se llame en caso de necesidad de destruir la actividad por lo que es el que se debería utilizar para salvar los datos necesarios si son muy importantes de manera persistente.

El método `onSaveInstancestate()` es llamado por el sistema antes que el método `onStop()` y permite guardar datos de la actividad en un objeto `Bundle` en formato de pares de valores, es decir, insertando un valor y una clave con la que recuperar dicho valor. Este objeto `Bundle` se pasará al método `onCreate()` y al método `onRestoreInstanceState()` para que puedan ser recuperados los valores en dichos métodos. Si la actividad es creada por primera vez este objeto `Bundle` es `null`.
Si no se sobrescribe el método `onSaveInstanceState()` en la actividad se llama al método `onSaveInstanceState()` por defecto de la clase `Activity` de la que hereda cualquier actividad. En este método se salva el estado de todas aquellas `Views` (botones, campos de texto editables, etc.) que tengan definido un id único. Sin embargo, no es seguro que el sistema llame a este método cuando una actividad es destruida, ya que hay casos en los que no es necesario guardar ningún estado. Por ejemplo cuando el usuario sale de la actividad pulsando el botón "Back", ya que se da por hecho que el usuario quería cerrar explícitamente dicha actividad o en caso de memoria extrema baja memoria en la que la actividad es cerrada de inmediato.

En la Figura 10 se puede ver un diagrama de cómo funciona este mecanismo:

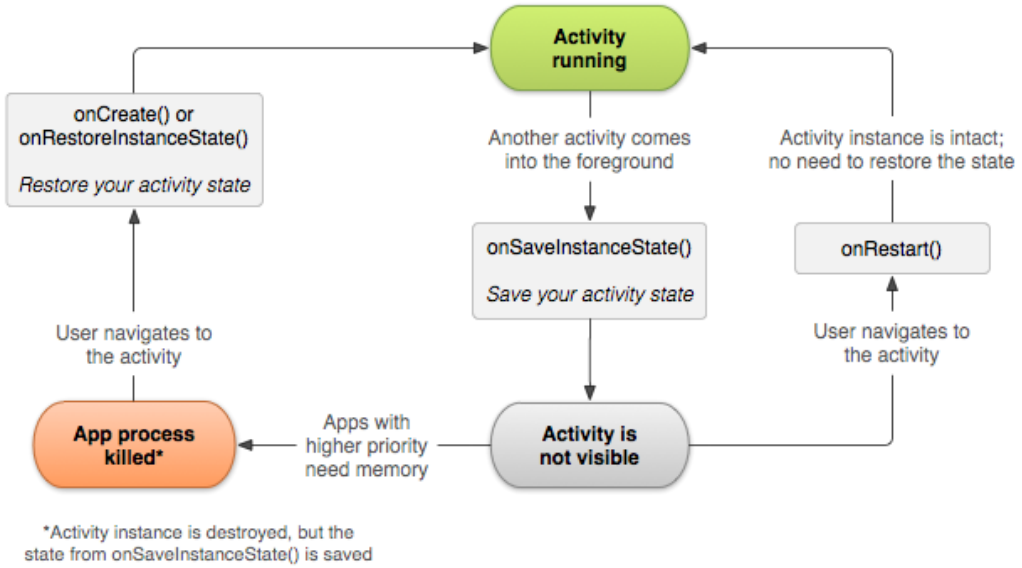

Figura 10 - Representación de cómo funciona el mecanismo para la recuperación del estado de una actividad. Extraído de [13]

2.3.10. Fragments

Un `Fragment` es una porción de interfaz de usuario de una actividad, o bien una porción de comportamiento que no tiene una interfaz de usuario asociada. Una actividad puede ser modulada en varios fragments formando una pantalla panelada, y cuyo panel (fragment) puede ser manejado independientemente de los demás.
Un fragment tiene su propio ciclo de vida, maneja sus propios eventos y pueden ser añadidos o eliminados mientras la actividad se está ejecutando. Todo fragment ha de estar embebido en una actividad, y el ciclo de vida de un fragment dependerá directamente del ciclo de vida de la actividad en la que esté embebido, por lo que si una actividad está pausada, todos sus fragments estarán pausados y cuando la actividad sea destruida también serán destruidos todos sus fragments embebidos. Pero mientras esté activa, sus fragments pueden estar en distintos estados de su propio ciclo de vida.

Fueron introducidos a partir de la versión 3.0 (nivel de API 11) de Android para dotar de mayor sencillez y dinamismo a la creación de interfaces para tablets. Un fragment puede ser reutilizado en distintas actividades. Ha de ser modular y reutilizable ya que esto permite que se puedan diseñar combinaciones de fragments para distintos tamaños, obteniendo versiones tanto para handset (teléfonos de dimensiones pequeñas que caben en una mano) como tablets.

En la siguiente figura se ve el ejemplo típico de uso de dos fragments, para tablets: uno con una lista a la izquierda, y a la derecha otro fragment con la información según lo que se haya seleccionado en la lista de la izquierda; en handset: por separado, primero la lista y cuando se pulsa sobre algún elemento se muestra la información. De este modo se puede aprovechar el espacio de la pantalla de ambos dispositivos de la mejor manera posible, como se puede observar en la Figura 11.

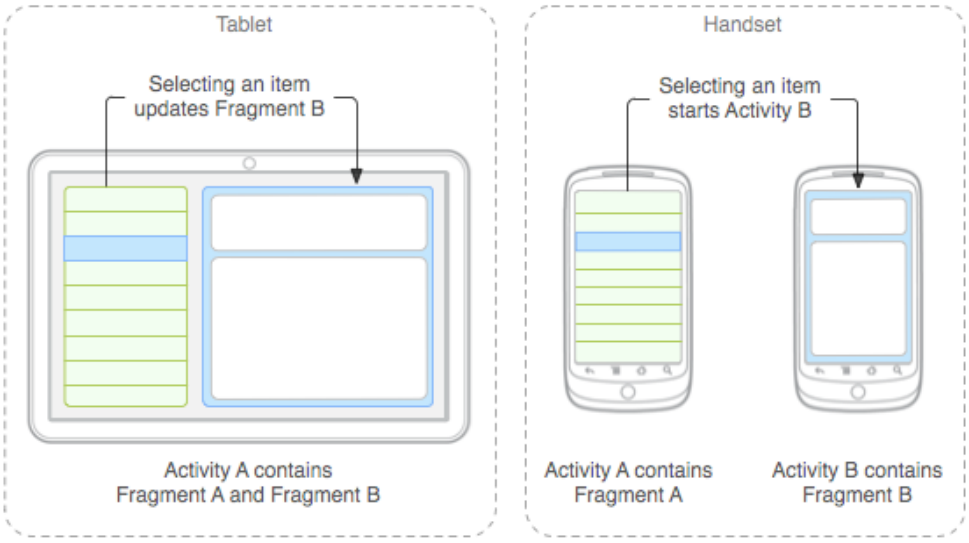

Figura 11 - Ejemplo de uso de fragments para tablet y handset. Extraído de [14]

Un fragment puede no tener interfaz de usuario y ser utilizado para llevar a cabo tareas en segundo plano. Si ha de tener interfaz gráfica, se activan los componentes gráficos por primera vez en el método `onCreateView()`, el cual devuelve la view raíz del layout del fragment.

2.3.11. Shared Preferences

Las `SharedPreferences` son un tipo de almacenamiento persistente de pares clave-valor. Permite almacenar tipos primitivos como strings, boolean, int, etc. Los pares clave-valor se caracterizan por almacenar un valor con una clave, y para recuperar dicho valor se accede a través de la clave. Al ser un sistema de almacenamiento persistente, la información almacenada estará disponible entre distintas sesiones aun cuando la aplicación sea destruida. Se pueden establecer opciones de privacidad en la escritura y/o lectura para que esta información sólo pueda ser accedida desde la aplicación que la creó.

2.3.12. Threads y AsyncTask

Cuando se ejecuta una aplicación el sistema crea un hilo (thread) llamado *main* o *UI Thread* para dicha aplicación. Todo el sistema de eventos y todo lo relativo a la interfaz gráfica, incluido el dibujado de la pantalla irá en dicho hilo.
Es importante que este hilo no quede bloqueado por operaciones de larga duración como puede ser una consulta a Internet, ya que al usuario le parecerá que la aplicación se ha quedado bloqueada y que no responde. De hecho, si el sistema detecta que el UI Thread de una aplicación se ha quedado bloqueado unos 5 segundos mostrará al usuario el mensaje de que la aplicación no responde y le permitirá cerrarla.

Debido a ello, las operaciones de larga duración se suelen realizar en hilos aparte denominados *worker threads*. Sin embargo, la interfaz gráfica sólo puede ser modificada por el UI Thread, por lo que una vez que la tarea de larga duración se termina, como el descargar una imagen de internet, a la hora de pintarla en la pantalla se ha de hacer dentro de los métodos `Activity.runOnUiThread()`, `View.post()` o `View.postDelayed()`. De este modo la estructura se hace compleja ya que desde el UI Thread se crea un worker thread que a su vez vuelve a llamar al UI Thread. Para ayudar a hacer este proceso más simple Android incorpora en sus librerías la clase `AsyncTask`.

Creando una subclase de `AsyncTask` evita dicha complejidad, realizando el trabajo que haría el worker thread en el método `doInBackground()`, que trabaja en un worker thread por debajo directamente, y permite publicar resultados en la interfaz de usuario a

través de los métodos `onPreExecute()`, `onPostExecute()` y `onProgressUpdate()`. Desde `doInBackground()` se puede llamar en cualquier momento al método `publishProgress()` que hará que el sistema ejecute el método `onProgressUpdate()`. Por último el método `doInBackground()` ha de devolver un valor, que será el que se le pase como parámetro al método `onPostExecute()`. La tarea puede ser cancelada desde cualquiera de los threads.

2.4. Desarrollo de una aplicación en Android

Existen diversas características y herramientas a tener en cuenta a la hora de desarrollar una aplicación en Android. Algunas de ellas se enuncian a continuación:

2.4.1. Tipos de dispositivos

Lo primero que hay que pensar es para qué dispositivos se va a desarrollar la aplicación. ¿Móvil? ¿Tablet? ¿Reloj inteligente?
Lo más común es que se vaya a utilizar en un móvil, pero en el último año se está viendo un gran incremento en el uso de tablets, o incluso algunos móviles de gran tamaño como el Samsung Galaxy Note (phablet). Por lo que es importante desarrollar la aplicación de modo que se vea bien y aprovechando el tamaño de cada pantalla adecuadamente en el mayor número de dispositivos posibles.

Hay varias maneras de afrontar este problema:

- Tener versiones distintas en su market de una misma aplicación según sea para un handset, tablet, e incluso distintos tamaños. Algunos desarrolladores prefieren utilizar este método y tener una aplicación dedicada a la tablet para aprovechar al máximo el tamaño de la pantalla de ésta sin complicarse con el código de manejar los tamaños en una misma aplicación. Sin embargo significa tener que estar manteniendo al menos dos aplicaciones distintas, con la mayoría del código duplicado, por lo que cuando se haga un cambio en el código, éste deberá hacerse en ambas aplicaciones, con la consiguiente pérdida de tiempo y por lo tanto, de dinero. Además, teniendo en cuenta la gran variedad existente en Android de tamaños, de densidades de pantalla, etc., sería inviable el mantener una aplicación distinta para cada uno de los tipos.

- Sin embargo en Android existen varios mecanismos para manejar los problemas que conlleva tener tantos tipos de dispositivos, no sólo la diferencia de tamaño entre un smartphone y una tablet, siendo el más importante de ellos el uso de recursos alternativos. El uso de recursos alternativos se ha explicado en mayor

profundidad en el apartado 2.3.3. Entre otras muchas utilidades, permiten tener unos recursos (imágenes, layouts, dimensions, styles, etc) distintos para diferentes tamaños de pantalla, con los que podemos crear distintas pantallas según sea un móvil o una tablet y el sistema se encargará de elegir el más óptimo según el dispositivo móvil en el que se esté ejecutando la aplicación.

- Existe otra opción relacionada estrechamente con la versión de Android en la que se esté realizando la aplicación, la cual se explica en mayor detalle en el siguiente apartado. Se trata del uso de fragments (disponibles a partir de la versión 3.0 o de forma más limitada en versiones utilizando la librería de compatibilidad). Para más información sobre fragments consulten la sección 2.3.9.

2.4.2. Versión de Android

Hay que decidir desde un primer momento a partir de qué versión de Android va a poder funcionar nuestra aplicación.
Hay varias opciones que se pueden tomar, en las cuales influye en gran medida para qué dispositivos está orientada la aplicación, cada una con sus pros y sus contras:

Desde la versión 3.0 en adelante se introducen importantes cambios en el API (Application Programming Interface) de Android. Principalmente se introduce el uso de fragments que permiten desarrollar tanto para móviles como para tablets de forma dinámica en una misma aplicación. Otro cambio importante es la existencia de la *action bar* para mejorar control del flujo del programa entre las distintas pantallas y eliminar la necesidad del botón físico de menú. Más cambios serían el uso de la interfaz holo que conlleva un gran cambio a nivel gráfico y de apariencia de las vistas más comunes como son los botones, cuadros de texto editables o spinners, o nuevos componentes como `ViewPager`, etc.

Para algunas de estas nuevas funcionalidades Google ha creado unas librerías de compatibilidad que permiten utilizar de forma algo más limitada estas nuevas características de las nuevas versiones en dispositivos con versiones más antiguas de Android. Sin embargo sigue habiendo mejoras que no están en la librería de compatibilidad, e intentar simular este comportamiento o interfaz nueva en las versiones viejas puede ser bastante costoso y hace que muchos desarrolladores elijan dar soporte a la versión 4.0 en adelante.

De igual manera que pasaba con el problema entre móviles/tablets, algunos desarrolladores deciden hacer dos versiones de una misma aplicación, para versiones anteriores a 4.0 y en adelante de 4.0, como es el caso por ejemplo de Gmail. Sin embargo, sigue acusando los mismos problemas de duplicidad de código y esfuerzo expuesto anteriormente, pero aún más acuciantes pues es mantener otro código bastante

diferente. Además, los usuarios prefieren una interfaz más moderna y que use los últimos elementos gráficos disponibles en Android aunque tengan un móvil con una versión antigua, y muchas veces es motivo de utilizar una determinada aplicación u otra similar. Existe una cierta complejidad como se verá en la sección 3.9 al querer tener una misma interfaz y funcionalidad para todas las versiones. Sin embargo, las librerías de compatibilidad ayudan en gran medida a reducir dicha complejidad y suele ser preferible tener una sola aplicación a mantener y una misma interfaz gráfica de usuario para todas las versiones de modo que sea más consistente.

2.4.3. Tipos de densidades de pantalla

Al igual que diferentes tipos de pantalla existen dispositivos que aún con el mismo tamaño de pantalla o similar, difieren mucho unos de otros por su densidad de pantalla. La densidad de pantalla se define como la cantidad de pixeles que hay en un área física de la pantalla del dispositivo. Android agrupa las pantallas en cuatro grandes grupos: baja densidad (ldpi), media (mdpi), alta (hdpi) y muy alta (xhdpi). Para solucionar este problema se utilizan de nuevo los recursos alternativos, teniendo imágenes con distintas resoluciones para los distintos tipos de densidad de pantalla.

Además las medidas utilizadas en la interfaz de usuario de la aplicación nunca deberían ser en píxeles directamente sino en dp (*density-independent pixel o* píxeles independientes de la densidad). El sistema se encarga en tiempo de ejecución de escalar en función de la densidad del dispositivo en el que se está ejecutando. 1dp equivale a 1 px (píxel) en una pantalla de densidad media, es decir 160 dpi (*dots per inch* o puntos por pulgada). La conversión por tanto entre dp y px de la pantalla sería de la siguiente forma: px = dp * (dpi / 160).

2.4.4. Diferentes idiomas

Otro beneficio en el uso de recursos alternativos en el desarrollo de una aplicación Android es el poder traducir de forma sencilla una aplicación de un idioma a otro. Para ello, todas las cadenas de caracteres que aparecen en el código, o al menos aquellas que se van a mostrar al usuario estarán definidas en el fichero strings.xml o ficheros afines que contengan `Strings` (cadenas de caracteres) o `StringArrays` (arrays de cadenas de caracteres).

Como ya se vio en la sección 2.3.3, se pueden utilizar distintos valores en los recursos de la aplicación en función de las diferentes necesidades, como son el tamaño de la pantalla (x-large, large, etc.), o en este caso el uso de carpetas de recursos distintas según el idioma que tiene seleccionado el usuario como idioma para el sistema del dispositivo.

Para ello, en la carpeta de recursos del sistema, se añaden tantas carpetas con el nombre values-XX, siendo XX el código del idioma, como idiomas se quieran tener disponibles.

Por ejemplo, si queremos el idioma español por defecto se tendrá el archivo strings.xml con las cadenas de caracteres en español en la carpeta values, y otro archivo strings.xml con los mismos nombres utilizados en strings.xml de la carpeta values, pero esta vez con los valores de las cadenas de caracteres en inglés, en una carpeta values-en.

Los códigos para los distintos idiomas se definen por los códigos del lenguaje ISO 639-1 [16] y opcionalmente por un código de región de dos caracteres, precedidos por una "r" minúscula del ISO 3166-1-alpha-2 [17].

2.4.5. El IDE Eclipse y el SDK Tools de Android

A la hora de desarrollar una aplicación en Android, lo más utilizado es el IDE (Integrated Development Enviroment) Eclipse junto con el plugin ADT (Android Developer Tools).
Hoy en día desde la página web de Android Developers se puede descargar un paquete denominado ADT Bundle[1] en el que vienen incluidas el IDE Eclipse y el ADT plugin ya configurados correctamente Además contiene otras herramientas como la última versión de Android, una imagen del sistema Android para el emulador o las herramientas del SDK. El SDK y sus herramientas contienen las librerías, y herramientas necesarias para programar y solucionar bugs en Android.
Se pueden descargar otras versiones (llamadas platforms) de Android desde la utilidad SDK Manager o crear todo tipo de terminales virtuales para utilizarlos en el emulador como se explica en más detalle en la siguiente sección desde el Android Virtual Device Manager.
El plugin ADT hace que las herramientas del SDK se integren en Eclipse pudiendo ejecutarlas desde dentro de Eclipse como en el caso del DDMS (Dalvik Debug Monitor Server), la vista de Eclipse en la que se pueden ver las trazas creadas por el desarrollador en la aplicación o los mensajes de error que lanza el sistema cuando una aplicación falla.

[1] El paquete con el ADT Bundle se puede descargar desde: http://developer.android.com/sdk/index.html

2.4.6. Desarrollar utilizando el emulador o un dispositivo móvil físico

El plugin ADT del SDK de Android para el IDE Eclipse provee de un emulador de dispositivos móviles. De este modo se puede probar la aplicación en el propio ordenador sin necesidad de tener que tener un dispositivo móvil físico.

El emulador es bastante configurable, pudiendo elegir entre otras cosas el tamaño de la pantalla y su densidad, la versión del sistema operativo Android que lo moverá, si tiene tarjeta de almacenamiento externo SD y de qué tamaño, la memoria del dispositivo... Además se pueden tener tantos terminales virtuales como se deseen.

La mayor potencia del emulador es que al poder emular distintos tamaños de pantalla y distintas versiones del sistema operativo, permite comprobar el aspecto que tendrá la aplicación en pantallas de distinto tamaño y comprobar si dicho aspecto es el adecuado. Además permite comprobar que realmente funciona del modo esperado para las versiones de Android en las que se quiere que se pueda ejecutar la aplicación, sin tener que comprar un dispositivo móvil físico con cada una de dichas características para probar.

Sin embargo el emulador tiene sus limitaciones. Los sensores no pueden ser probados fácilmente. Existen librerías externas que ayudan a probar los sensores pero son complicadas de utilizar y los resultados distan de los obtenidos con un dispositivo móvil físico. Además es muy lento y puede llegar a ser exasperante cuando se realizan muchas pruebas seguidas de pequeños cambios en la interfaz gráfica.

Para probar la aplicación en un dispositivo móvil físico se ha de conectar al ordenador y tener instalados los *drivers* de dicho dispositivo móvil en el ordenador. Eclipse detectará el dispositivo móvil y cuando se le dé a ejecutar se lanzará directamente en el dispositivo móvil, o mostrará un menú para elegir sobre cual dispositivo ejecutar la aplicación si hay varios conectados al ordenador o uno o varios emuladores iniciados.

Lo más recomendable es probar la aplicación que se esté desarrollando en un terminal físico, ya que se puede probar a la velocidad normal que un usuario final tendrá en su dispositivo pudiendo diseñar una experiencia de uso mejor. Además permite probar sensores y otras características limitadas del emulador de forma sencilla. También es útil utilizar el emulador para pruebas en otras pantallas y diferentes versiones de Android, pero sólo como comprobación de que todo está correcto para diferentes dispositivos móviles, y no como herramienta de desarrollo habitual.

2.5. Web services

Se le denomina web service o servicio web a un conjunto de aplicaciones o tecnologías que sirven para intercambiar datos entre distintas aplicaciones de distintas plataformas a través de la Web. Dicha interoperabilidad se consigue mediante la utilización de estándares abiertos como XML, SOAP o HTTP.

En el caso de la aplicación de telefonía corporativa de la UC3M se han utilizado diversos web services para la obtención de datos, algunos de los cuales ya están implementados y en funcionamiento y otros aún no.

Estas solicitudes de consulta a los web service que se utilizan en la aplicación de este proyecto se realizan a través de peticiones POST utilizando el protocolo seguro HTTPS. Las peticiones POST permiten enviar la información sensible como los credenciales del usuario (usuario y contraseña) en una entidad que se añade al cuerpo de la petición de forma cifrada en lugar de enviarla en la propia dirección del enlace de la petición. El protocolo HTTPS asegura que la información se envía cifrada.

2.5.1. Web service de facturación y tarifas de telefonía

Para la obtención de los datos de consumo y tarifas es necesario obtener los datos de cada usuario, excepto para la consulta de todas las tarifas de telefonía actuales disponibles que serán iguales para todos los usuarios. Estos web services serán creados por la empresa externa que ha contratado la universidad para tratar todos los datos de tarificación. Por ahora se utilizan en la aplicación archivos XML con datos falseados sirviendo de ejemplo de cómo han de ser los archivos reales de las respuestas que se obtendrán de dichos web service.

Desde la aplicación se enviará el tipo de consulta que se quiere hacer al web service y el usuario para el cual se requiere dicha información. El web service devolverá la información en un archivo XML del cual la aplicación obtendrá la información necesaria para presentar al usuario.

2.5.2. Web service para la validación en LDAP

Todo el módulo de consulta de consumo y tarifas necesita que el usuario se haya autenticado contra el LDAP de la universidad para comprobar que es miembro de la misma. Un LDAP (Lightweight Directory Access Protocol o Protocolo de acceso ligero a directorio) es un estándar abierto para el acceso y actualización de un directorio, un tipo de base de datos que permite realizar búsquedas concretas, sencillas y rápidas.

El web service que se comunica con el LDAP de la UC3M y valida las credenciales introducidas por el usuario en la aplicación ha sido creado por un desarrollador de aplicaciones web que trabaja en la UC3M. Este web service ya ha sido creado y se encuentra en funcionamiento en la aplicación.

2.6. Buenas prácticas en el desarrollo de aplicaciones móviles

A continuación se exponen una serie de buenas prácticas a la hora de desarrollar una aplicación de software, no sólo para dispositivos móviles.

2.6.1. Sistema de control de versiones (SCV)

Un sistema de control de versiones (SCV) es una herramienta que gestiona los cambios realizados a los elementos de un proyecto o producto. Estos sistemas facilitan la administración de las distintas versiones, o también llamadas revisiones, de cada producto desarrollado, entendiéndose como versión al estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación.

A continuación se enumeran las funcionalidades que un SCV debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar. Por ejemplo, para un proyecto en Android se almacenarán el código de la aplicación, imágenes, documentación...).

- Posibilidad de realizar cambios sobre los elementos almacenados Por ejemplo: modificaciones parciales, añadir, borrar, renombrar o mover elementos.

- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

Aunque no es estrictamente necesario, suele ser muy útil la generación de informes con los cambios introducidos entre dos versiones, informes de estado, marcado con nombre identificativo de la versión de un conjunto de ficheros, etc.

Un SCV necesita de un repositorio en el que alojar, en el caso de una aplicación Android el proyecto con el código y los recursos utilizados. Un repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor.

Cada desarrollador tiene siempre una copia local de los ficheros del repositorio, y los cambios los realiza en esa copia local. Esto permite que los desarrolladores puedan trabajar en paralelo sobre el mismo proyecto. Cuando un desarrollador en cuestión quiere aplicar los cambios que ha hecho en su copia local al repositorio (acción denominada normalmente *commit*), según el esquema del sistema de versiones que se esté utilizando se hará de una manera u otra. Los posibles esquemas son:

- Sistema exclusivo: En este esquema de funcionamiento, cuando un desarrollador quiere realizar cambios sobre un fichero del repositorio, se le comunica al repositorio su intención y éste se encarga de bloquear que ningún otro desarrollador pueda realizar cambios en dicho elemento. Cuando se termina de modificar el elemento se comparten dichas modificaciones con el resto de desarrolladores y por último se libera el elemento para que los demás desarrolladores puedan realizar modificaciones si lo necesitan.
  Este modo de funcionamiento es el que usa por ejemplo SourceSafe[2]. Otros sistemas de control de versiones, por ejemplo Subversion, aunque no obligan a usar este sistema, disponen de mecanismos que permiten implementarlo. Se hablará más en detalle de Subversion al final de esta sección.

- Sistema colaborativo: En este esquema cuando el desarrollador quiere aplicar sus modificaciones en el código del repositorio, primero debe actualizar su copia local con la última versión disponible en el repositorio de todos los ficheros del proyecto. A este proceso normalmente se le denomina *update*. Si otro desarrollador ha subido sus cambios previamente al repositorio, pueden producirse diferentes situaciones:

  - Se realiza un *merge* automáticamente. Cuando los distintos desarrolladores no han estado modificando un mismo fichero del proyecto, el SCV integra en los ficheros de tu copia local las modificaciones realizadas en cada uno de los ficheros en los que hubo cambios. Si los cambios son en el mismo fichero, excepto que cambien las mismas líneas de código la integración será automática.

  - Hay que resolver los conflictos manualmente. Cuando los cambios son en las mismas líneas de un mismo fichero se produce un conflicto. El SCV lanza un error informando de que se encontraron conflictos que debe resolver el desarrollador. Para ello muestra al usuario el código de su copia local y la versión existente en el repositorio indicando qué líneas de código está produciendo el conflicto, para que elija qué versión quedarse.

[2] SourceSafe: http://msdn.microsoft.com/es-es/library/3h0544kx(v=vs.80).aspx

Un vez realizados todos los *merge* automáticos y resueltos todos los conflictos, ya estará sincronizada la copia local con la versión existente en el repositorio, por lo que podrá realizarse el *commit* con las modificaciones que el desarrollador previamente hizo en local.

Como se indicó anteriormente, para sincronizar tu copia local con la última versión disponible en el repositorio se ha de ejecutar el comando *update* y resolver los posibles conflictos como se indicó en los párrafos anteriores.

Según el sistema utilizado para el almacenamiento de los datos en un sistema de control de versiones podemos diferenciar dos arquitecturas distintas:

- Centralizados: existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones importantes necesitan la aprobación del responsable. Algunos ejemplos son CVS[3] y Subversion[4].

- Distribuidos: Cada desarrollador tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Además es frecuente el uso de un repositorio central, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git[5] y Mercurial[6].

Aunque un SCV tiene más funcionalidades muy útiles como son las *branches* (ramas) o los *tags* (etiquetas), sólo se va a hablar de estas dos muy brevemente ya que no han sido utilizadas en el desarrollo de este proyecto fin de carrera.

- Branch o rama: Se crea una copia que evoluciona de forma independiente a la versión principal. Más adelante se puede hacer un *merge* con la rama principal, por lo que se suelen utilizar como rama de pruebas de nuevas funcionalidades, de corrección de errores...

- Tag o etiqueta: Se crea una tag en el repositorio cuando se llega a una versión importante estable de la aplicación por si se necesita volver a ésta en un momento dado. Una vez creado el tag no se le añaden nuevas modificaciones sobre éste.

Utilizar un sistema de versiones aporta grandes ayudas a la hora de desarrollar cualquier aplicación, como son:

[3] CVS: http://cvs.nongnu.org/
[4] Subversion (SVN): http://subversion.apache.org/
[5] Git: http://git-scm.com/
[6] Mercurial: http://mercurial.selenic.com/

- Tener una copia de seguridad que no está alojada en el ordenador en el que se está desarrollando la aplicación.

- La recuperación de versiones estables en caso de que se haya modificado algo en la nueva versión que produzca un error y no se consiga localizar dicho error, o de que un cambio en la última versión produzca un error colateral y sea preferible regresar a la versión anterior.

- Comparar versiones entre sí, para localizar los cambios hechos en cada versión rápidamente.

- Trabajar de forma colaborativa más fácilmente sobre el mismo código e integrar de forma automatizada siempre que sea posible los cambios realizados por cada desarrollador.

- Aunque sea sólo una persona la que esté desarrollando la aplicación, es útil si se trabaja desde diversos puestos de trabajo, o desde distintos ordenadores.

A continuación se hablará sobre dos de los sistemas de control de versiones más utilizados hoy en día: Subversion y Git:

- Subversion (SVN): Es un sistema de control de versiones de software libre bajo una licencia de tipo Apache/BSD, lanzado en el año 2000.
  Sigue un esquema centralizado. Sólo se envían las diferencias tanto al actualizar la copia local como al subir los cambios al repositorio central.

- Git: Sistema de control de versiones de software libre bajo licencia GNU General Public License versión 2, lanzado en el año 2005.
  Sigue un sistema distribuido. Fue pensado para mantener la eficiencia y velocidad en las transacciones de datos tanto en proyectos pequeños como en proyectos muy grandes y con muchos colaboradores. Mejora mucho en este aspecto a subversion, mucho más lento en la operación de crear la copia local inicial, además de más proclive a fallos y corrupción de datos al realizar las operaciones de *merge* entre ramas.

2.6.2. Testing sistemático

El testing, o pruebas de software, se encarga de mejorar la calidad del código a través de la creación de distintas pruebas que se llevan a cabo sobre el código. Permite obtener un código más robusto y menos susceptible a que por posibles cambios futuros en el código se rompan otras funcionalidades que aparentemente funcionaban bien previamente.

El testing sistemático consiste en crear un paquete de pruebas automáticas que exponen el código a que de la respuesta esperada dadas unas premisas o valores de entrada. Permiten probar así que esa pieza de código funciona tal y como está previsto en todos los diferentes escenarios que al programador se le ocurra, de forma rápida y automática en lugar de tener que ir probando escenario a escenario distinto. Esto permite que los paquetes de pruebas puedan ser pasados a menudo sin la problemática de que alguno no se realice por descuido o porque sean pesados de realizar si hay que cambiar partes del código o cambiar mucho la configuración del dispositivo móvil para cada una de las pruebas. Esto conlleva un gran ahorro de tiempo en el testeo de cada funcionalidad.

Un test unitario es aquel que se encarga de probar unidades pequeñas de código. La granularidad del ámbito del test es variable, desde métodos, un conjunto de ellos, clases… Se intenta que sean probados de forma aislada de otros elementos como bases de datos o entrada de información desde internet. Estos datos serán emulados para que los test se centren simplemente en la funcionalidad de la pieza de código que se está probando.

Un test de sistema prueba la integración entre varias partes del sistema o aplicación, utilizando los mismos puntos de entrada que en la aplicación real utilizada por el usuario, incluida la interfaz gráfica. De este modo se puede probar que dada una interacción del usuario con la interfaz gráfica la aplicación reacciona de la forma esperada, pero haciéndolo de forma automatizada con todos los posibles escenarios que crea convenientes el programador.

En Android existe una serie de clases específicas para realizar test sin necesidad de otras librerías externas, basadas en JUnit 3. Las clases más importantes son `ActivityInstrumentationTestCase2`, `ActivityUnitTestCase`, `ProviderTestCase2` (para content provider) y `ServiceTestCase` (para service).

Además, aunque todo se puede hacer con las librerías que Android provee para los test, se ha utilizado la librería externa Robotium para la creación de test de forma más sencilla, rápida y legible.

- **Robotium**

Robotium [55] es un framework para crear test automatizados sobre Android de forma fácil y rápida, sobre todo para probar la interfaz gráfica, tarea mucho más pesada y compleja utilizando sólo las librerías de test de Android.

Permite realizar tanto tests white-box (caja blanca) para los que es necesario conocer en profundidad el código a probar, como de black-box (caja negra) para los cuales no es necesario conocer en detalle el código.
Con Robotium se pueden hacer test de black-box en Android indicando tan sólo que ha de buscar cierto texto en la pantalla visible en ese momento, o pulsar el botón con determinado texto, sin conocer el nombre de la variable o el id de ese botón. Otros posible ejemplo sería pulsar sobre la tercera opción de una lista en concreto y haciendo una asertación sobre la respuesta esperada a esa interacción con la interfaz gráfica. De este modo se pueden crear test complejos de forma rápida y sencilla.

2.6.3. Patrones de diseño de software

Los patrones de diseño software son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Dan soluciones que han sido altamente comprobadas su efectividad resolviendo problemas similares y que además son reutilizables en distintas circunstancias del mismo problema. Pretenden de este modo evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente. Sin embargo no se impone su uso sobre otras alternativas de diseño, y de hecho, el forzar el uso de patrones o abusar de ello puede ser un error.

Una posible clasificación de los patrones según el propósito para el que han sido definidos es:

- Patrones de creación: Resuelven problemas relacionados con la creación de instancias de objetos. Algunos ejemplos de patrones de creación son el *Singleton*, y el *Factory Method*.[7]

- Patrones estructurales: Solucionan problemas de composición (agregación) de clases y objetos. Algunos ejemplos de patrones estructurales son el *Adapter* y el *Composite*.

- Patrones de comportamiento: Ofrecen soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan. Algunos ejemplos de patrones de comportamientos son el *Iterator* y el *Observer*.

[7] Para más información sobre distintos patrones de diseño consultar la siguiente página web: http://patronesdediseno.net16.net/index.html

- Patrón de interacción: Son patrones para crear interfaces de usuario. Algunos ejemplos de patrones de interacción para Android en concreto son el *Data drill down* o las *Expandable list*[8].

En el desarrollo de la aplicación de telefonía corporativa de la UC3M se hace uso del patrón ***Singleton***. *Singleton e*s un patrón de creación que resuelve la necesidad de tener sólo una única instancia de un determinada clase, proveyendo de un punto de acceso global a dicha instancia. Esto se consigue gracias a que es la propia clase la responsable de crear esa única instancia, (declarando el constructor de la clase como privado) y a que se permite el acceso global a dicha instancia mediante un método de clase (usando la palabra reservada `static` en Android).
Además se utilizan diversos patrones de interacción en la aplicación tales como Data drill down cuando es necesario mostrar mucha información.

2.6.4. Refactorización de código

Se denomina refactorización de código a la técnica de reestructurar, modificar y re-escribir código de manera que externa siga teniendo el mismo comportamiento, pero que implique alguna ventaja en su estructura interna.

Se suele utilizar para "limpiar" código muerto que finalmente no se usa o que antes se usaba y ya no pero que se dejó por si acaso, eliminar código duplicado, obtener código más reutilizable, más sencillo de entender y de mantener, más compacto o más eficiente. Estos objetivos están relacionados con la calidad del código (cómo lo hace) y no con su funcionalidad (qué hace).

Dado que la refactorización es parte del mantenimiento del código y que no debe cambiar su funcionalidad, no se considera refactorizar el arreglar errores o añadir funcionalidad extra.

La refactorización se suele utilizar junto al uso de test, para comprobar que después de la refactorización la aplicación sigue funcionando correctamente en todos los casos y que no se ha creado ningún error colateral en la aplicación.

La refactorización puede ser una experiencia tediosa si se hace de forma manual, pero hoy en día existen herramientas que hacen gran parte de este trabajo de forma automática. El IDE Eclipse permite realizar refactorizaciones automáticas, actualizando las referencias necesarias, de las refactorizaciones más comunes, como por ejemplo

[8] Para más información sobre patrones de diseño de interacción en Android consultar la siguiente página web: http://www.androidpatterns.com/

mover una clase de un paquete a otro, renombrar una clase o método, o seleccionando el código que se desee extraer un nuevo método de ese código para mayor modularidad o si se reutiliza en varias ocasiones.

2.6.5. Buenas prácticas en la codificación

Dado que la aplicación de telefonía corporativa de la UC3M es una aplicación para uso real por los miembros de la UC3M va a tener que ser mantenida a lo largo de los años y puede que necesite ser modificada por otros desarrolladores. Por ello es necesario tener en cuenta otras buenas prácticas a la hora de escribir el código, para que sea más legible y todo lo sencillo que pueda. De este modo, en caso de que un nuevo desarrollador tenga que tratar con la aplicación sin conocer el código previamente se pueda hacer al código rápidamente o incluso el mismo desarrollador que realizó la aplicación pero cierto tiempo después. A continuación se enumeran algunas de ellas:

- El idioma de las variables, métodos, clases, comentarios, etc. debe ser siempre el mismo siempre, y preferiblemente en inglés, ya que puede que la persona que ha de tratar con el código no sepa expresarse bien en el idioma materno de los otros desarrolladores, por lo que es preferible utilizar idioma más universal, que hoy en día es el inglés.

- Los nombres de las variables, los métodos, las clases, etc. deben ser los más descriptivos posibles de la funcionalidad para los que se utilizan. Por ejemplo, no utilizar una variable para tiempo *t*, sino *time*.

- Indentar correctamente el código para una mayor legibilidad. De igual modo utilizar siempre el mismo método de abrir y cerrar las llaves. O bien siempre la llave inicial en la misma línea del nombre del método, o bien siempre debajo de éste, pero no mezcladas ambas aproximaciones.

- Comentar el código allí donde sea necesario, describiendo toda aquella funcionalidad que no quede fácilmente clara a priori, pero sin llegar a describir cosas que ya quedan claras por sí mismas para evitar una pérdida de tiempo.

- Utilización de la convención de nombres *CamelCase* para el código en Android al igual que se utiliza en Java. Si el nombre de una variable o método es compuesto se utiliza la primera palabra en minúscula excepto que sea un acrónimo y la primera letra de la siguiente palabra en mayúscula, como por ejemplo myVariable. En el caso de una clase las primeras letras de cada palabra irán en mayúscula, como por ejemplo MyClass. Las constantes irán en mayúsculas separando las palabras con una barra baja (“_”), como por ejemplo MY_CONSTANT. Los acrónimos pueden ir tanto en mayúsculas como en minúsculas, como por ejemplo myXMLFile o myXmlFile.

- En archivos XML se tratará de utilizar la convención *snake_case*. Todas las letras han de ser minúsculas y se usa una barra baja (“_”) para la separación entre palabras.

2.6.6. Consideraciones a tener en cuenta cuando se desarrolla para dispositivos móviles

Las buenas prácticas mencionadas anteriormente son válidas para cualquier desarrollo de una aplicación, no sólo para móviles. En esta sección se hablará de cuestiones a tener en cuenta cuando la aplicación sea desarrollada para un dispositivo móvil.

En primer lugar hay que tener en cuenta las limitaciones del hardware del dispositivo. Aunque cada día se consigan dispositivos con un mayor hardware sigue siendo más limitado que un PC (Personal Computer u ordenador de uso personal).
Las pantallas son más pequeñas por lo que habrá que cuidar la cantidad de información que se presenta en cada una de ellas. Además los usuarios de móviles suelen echar vistazos rápidos a la pantalla por lo que la información deberá presentarse de forma clara y sin abusar de la cantidad de información que acabaría agobiando al usuario y creando una mala experiencia de uso.

Por otro lado, la memoria del dispositivo es limitada. Se ha de tener en cuenta que se van a estar ejecutando otras aplicaciones en el dispositivo y diversos procesos en segundo plano de otras tantas. Se ha de tener un buen diseño de la aplicación que tenga en cuenta los cambios entre el paso de primer plano a segundo y viceversa, y contemplar los casos en que alguno de los componentes de la aplicación que estén en segundo plano pueden ser destruidos (actividades y services) y cómo han de comportarse en dichos casos para que vuelvan al estado en el que estuvieran cuando el usuario desee volver a ellos.

Otra práctica necesaria es liberar recursos que no se estén utilizando si la aplicación no está en primer plano, como los sensores en un videojuego que está pausado o el GPS, que conllevan un gran gasto en memoria y batería al dispositivo.

El almacenamiento interno del dispositivo también es limitado. Se han de realizar aplicaciones que ocupen lo menos posible puesto que hay muchas otras aplicaciones ocupando su propio espacio.
Muchos de los dispositivos móviles permiten tener tarjetas de almacenamiento externas, por lo que es interesante permitir al usuario que almacene la aplicación en la memoria externa, o mover parte de los archivos grandes de la aplicación a la memoria externa. No en todas las aplicaciones se ha de permitir esto, ya que si por ejemplo el usuario conecta el móvil a un ordenador y lo utiliza como almacenamiento externo, la tarjeta SD quedará bloqueada para el dispositivo móvil, similar a si la hubiera quitado del todo. Por ello,

aplicaciones en tiempo real que necesitan estar activas siempre, como la del reloj, o aplicaciones de mensajería no suelen permitir esta opción.

Se ha de poner atención también a que una aplicación no quite el foco y se ponga en primer plano sobre la que estaba utilizando el usuario, al menos sin la opción de desactivar este comportamiento tal y como hacen algunas aplicaciones como Whatsapp o ChatOn. Existen diversas maneras de notificar al usuario sin llevar a cabo este comportamiento, siendo las más comunes las notificaciones en la barra de estado, y/o la vibración del dispositivo.

Se ha de desarrollar pensando en que las conexiones a internet pueden ser lentas, por lo que deberían realizarse en segundo plano e informar al usuario de lo que está ocurriendo mediante un mensaje en la pantalla o la típica imagen de una barra con el progreso horizontal o circular.

Se ha de tener en cuenta que existe una gran diversidad de pantallas (densidad de pantalla, tamaño...) y diversidad de modos de entrada (pantalla táctil, teclado, *trackball*, acelerómetro, *joystick*...).

La aplicación ha de ser eficiente y responder de forma rápida a la interacción con el usuario, e informarle cuando se estén llevando a cabo operaciones lentas. Android considera que si una aplicación tarda más de cinco segundo en responder, la aplicación se ha quedado colgada y da al usuario la opción de detener la aplicación.

Por último hay que tener en cuenta que algunos servicios pueden llevar asociados costes para el usuario, como el envío de SMS, el acceso a Internet, realizar llamadas etc. Se ha de informar al usuario de ello, aunque al instalar la aplicación ya aceptase dichos permisos.

Capítulo 3

Desarrollo de la aplicación de telefonía corporativa de la UC3M

3.1. Introducción

La aplicación de telefonía corporativa de la UC3M ha sido pensada para que los empleados de la universidad puedan acceder de forma fácil, cómoda y desde cualquier lugar a los servicios más demandados del área de telefonía de la UC3M. Estos servicios son principalmente la consulta de la facturación y tarifas telefónicas, la consulta de las guías de configuración de sus smartphones o tablets corporativos, la notificación de incidencias al servicio encargado en la universidad y la activación y la configuración del servicio de telefonía IP.

En este capítulo se describe en detalle el desarrollo de la aplicación, cada una de sus funcionalidades en detalle y las tecnologías implicadas en su desarrollo y uso.

3.2. Requisitos del proyecto

Los requisitos de la aplicación, como ocurre en casi todo proyecto real, han ido cambiando, añadiendo nuevos o eliminando otros durante el desarrollo del proyecto. El modo de trabajar en los requisitos con los distintos clientes ha sido a través de mockups en los primeros meses y en pequeñas demos de la aplicación para mostrar los avances sobre los que los clientes daban su visto bueno o cambiaban algunos requisitos.

En primer lugar, la responsable del servicio de telefonía de la UC3M como primer cliente creó un mockup con algunas de las pantallas de la aplicación y la información que quería que se mostrase, además de explicar la funcionalidad de cada una de esas pantallas, lo que constituye la base de los requisitos del proyecto. En el diseño de algunas de las pantallas del mockup también colaboró la autora de la presente memoria directamente sugiriendo métodos de navegación a través de mucha información o en otros aspectos gráficos mostrando los componentes de Android con los que se suele trabajar en otras aplicaciones Android. La cuarta y quinta versión del mockup de la aplicación lo realizó la diseñadora gráfica con los iconos creados propiamente para la aplicación y dando una aproximación más cercana a la interfaz de usuario final de la aplicación.

Algunas de las funcionalidades como el asesor de tarifas no fueron añadidas desde un principio si no varios meses después, por lo que, junto con algunas de las pantallas que no se habían diseñado previamente, no se incluyeron en ninguna de las versiones de mockups. El diseño de estas pantallas fue ideado por la programadora directamente y validado por la responsable del servicio de telefonía y posteriormente por los asesores de contenido de la UC3M.

A continuación se enumera la funcionalidad requerida para la aplicación en su fase final:

Los servicios ofrecidos se engloban en tres grandes grupos:

- Consulta de información sobre la facturación telefónica y de las tarifas de los empleados de la universidad.

- Guías de configuración de los smartphones y tablets homologados por la universidad y soporte.

- Información y activación del servicio de telefonía IP de la universidad.

Para la consulta de los datos relacionados con la telefonía corporativa es necesario que el usuario se valide con sus datos de la universidad en la aplicación. En esta área podrá acceder a distinta información relacionada con su telefonía corporativa, tanto para teléfonos fijos como móviles corporativos si dispone de uno o varios, como en el caso de tener un despacho en un campus y otro en otro. Concretamente se podrá acceder a la siguiente información:

- Consulta de su perfil, es decir, sus permisos de llamadas (nacional, móviles, internacional...) y sus tarifas actuales contratadas con el operador de telefonía que da servicio a la universidad.

- Consulta del consumo del usuario, ordenado por periodos. Como hoy en día el operador de telefonía contratado por la universidad manda la información de la facturación telefónica de la universidad por meses, los períodos son los distintos meses del año.

- Consulta del consumo de grupo, sólo disponible para responsables de departamento. En esta opción el usuario podrá consultar el consumo telefónico de todos los miembros de su departamento ordenado por orgánica y periodo.

- Un asesor de tarifas, que a través de distintas opciones según el uso que el usuario dé a su teléfono le indicará cual es la tarifa más adecuada de las ofertadas por el operador de telefonía contratado por la universidad para los miembros de la universidad.

- Y por último se ofrece el poder consultar las tarifas ofertadas por el operador de telefonía contratado por la universidad para los miembros de la universidad, mostradas por la necesidad del usuario: consumo de voz, consumo de datos y roaming.

El segundo bloque contiene información sobre la configuración del smartphone o tablet según en qué tipo de dispositivo esté ejecutándose la aplicación. Muestran la información más relevante contenida en las guías de configuración que realiza el área de telefonía de cada terminal homologado por la universidad. Algunos ejemplos serían la configuración para conectarse a la red Wi-Fi Eduroam, cómo configurar la cuenta de correo de la universidad o cómo conectarse a la VPN (Virtual Private Network o red privada virtual) de la universidad.
Además contiene un resumen de la normativa del servicio de telefonía de la universidad, y una guía de buenas prácticas cuando utilice su teléfono fijo o móvil corporativo.
Por último dispone de la opción de llamar al CASO (el Centro de Atención y Soporte de la Universidad Carlos III de Madrid) para poner una incidencia por un problema relacionado con la universidad.

El último bloque informa sobre qué es el servicio de telefonía IP de la universidad, dando la opción de activarlo y descargar la aplicación para su uso en el dispositivo móvil que esté utilizando el usuario. Por último se ofrece un resumen de la guía para configurar dicha aplicación con la información de su cuenta de VoIP (*Voice over IP* o voz sobre IP) de la universidad.

Otros requisitos fuera de la funcionalidad específica de la aplicación son:

- No se almacena en ningún caso la contraseña del usuario en la aplicación. Mientras se esté utilizando la aplicación el usuario mantendrá la sesión iniciada a menos que decida cerrarla utilizando el botón disponible para ello. Cuando el usuario no utilice la aplicación y el sistema la destruya, el usuario ha de volver a introducir la contraseña de nuevo.

- La información sensible de las credenciales del usuario han de enviarse por una conexión segura y cifrada a través de una petición POST.

- El menú de “Consumo de Grupo/Área” sólo será visible para aquellos usuarios que tengan los permisos necesarios. La comprobación de dichos permisos se realizará a través de una petición a un web service. Dicho web service estará al cargo de la empresa externa que provee los datos de tarificación y tarifas.

- Crear los prototipos de respuesta de los web service en los que se basará la empresa contratada externamente para manejar los datos de la facturación de la universidad y que ha de crear los web service atendiendo a dichos prototipos.

- El diseño de la interfaz ha de mantener la imagen corporativa creada previamente para las distintas aplicaciones ya existentes de la UC3M. Para ello, se colaborará con una diseñadora gráfica que se encargará de realizar los iconos de los menús y elegir algunos de los colores de la aplicación para que se integren con el del resto de aplicaciones Android de la UC3M.

- Crear un código comentado allá donde fuera necesario y mantenible fácilmente por otros desarrolladores que no tuvieran un conocimiento previo de la aplicación.

- Crear una interfaz clara e intuitiva.

- Crear una aplicación que funcione en, al menos, todos los dispositivos móviles Android homologados por la UC3M actualmente: HTC Desire HD, Galaxy S2 y Galaxy Tab. En el caso de las tablets, crear una interfaz que aproveche el tamaño de la pantalla.

En la Figura 12 se muestra algunas imágenes que componían la segunda versión del mockup, en la Figura 13 algunas de la tercera versión y en la Figura 14 algunas de las imágenes de la quinta y última versión del mockup. Como se observará más adelante en la sección 3.6 en la que se muestran imágenes de la última versión de la aplicación, casi todas las pantallas del área de “Facturación y Tarifas” han tenido que ser modificadas. Se ha tenido que utilizar un diseño basado en pestañas ya que no se había pensado realmente el volumen de datos que podría tener un usuario cuando se hicieron estos primeros diseños entre otros cambios.

Figura 12 - Imágenes pertenecientes a la segunda versión del mockup de la aplicación

Figura 13 - Imágenes pertenecientes a la tercera versión del mockup de la aplicación

Figura 14 - Imágenes pertenecientes a la quinta versión del mockup de la aplicación

3.3. Diseño de la interfaz gráfica

Dado que la aplicación de telefonía corporativa de la UC3M es una aplicación oficial de la universidad que ha de integrarse con el resto de aplicaciones creadas anteriormente ha de mantener la imagen corporativa de la UC3M. Esto se refleja por ejemplo en tener que usar una cabecera similar a las utilizadas en otras aplicaciones Android corporativas de la universidad. En la Figura 15 se puede ver esta cabecera:

Figura 15 - Cabecera de la aplicación

El diseño de la interfaz gráfica se ha realizado en distintos pasos. Lo primero fue crear en el departamento de telefonía unos mockups de lo que sería la aplicación. Los mockups son prototipos del diseño creados en papel o mediante aplicaciones informáticas de diseño gráfico. Estos mockups se mostraban a los asesores de contenidos de la universidad, que para este proyecto han ejercido de cliente junto a los responsables del departamento de telefonía ya que son los que debían validar la aplicación, y comprobar que la interfaz gráfica cumplía con la imagen corporativa de la UC3M.

El diseño de los iconos de los menús al igual que los colores utilizados en la aplicación fue creado por una diseñadora gráfica que trabaja para la universidad. Creó también una versión final de los mockups tras diversas conversaciones con ella para adaptar su diseño a un diseño en el que confluyera la imagen corporativa de la UC3M con el diseño de aplicaciones Android.

En cualquier caso, la aplicación finalmente difiere en algunas pantallas con el diseño creado en los mockups ya que a la hora de implementarlos se vio que debido a la posible gran cantidad de datos en algunas de las pantallas había que modificar el diseño inicial. También hubo que diseñar algunas pantallas que no estaban pensadas en el inicio del proyecto cuando se crearon las distintas versiones de los mockups como el asesor de tarifas.

Los iconos que se utilizan en la aplicación son los propios de Android para que sea más intuitiva la aplicación para los usuarios acostumbrados a Android. Los iconos creados por la diseñadora son aquellos utilizados en las listas de los menús como los de la Figura 16:

Figura 16 - Algunos iconos de los menús

3.4. Tecnologías utilizadas en el desarrollo

A la hora de desarrollar la aplicación de telefonía corporativa de la UC3M se ha hecho uso de las tecnologías que se describen a continuación:

- **ADT Bundle (Eclipse IDE + ADT Plugin)**

Desde la web de Android Developers facilitan el uso del IDE Eclipse con el plugin del ADT (Android Developer Tools) ya integrado. El uso de Eclipse con dicho plugin permite muchas opciones útiles a la hora de desarrollar en Android como son el emulador de terminales reales, el DDMS para ver los mensajes de error y los logs creados por el programador o el editor gráfico de interfaces de usuario. Eclipse además proporciona ventajas como mostrar los errores de codificación en tiempo real (por ejemplo si falta un ";" o si estás pasando parámetros no válidos a un método), autocompletado, refactorizaciones automáticas más comunes entre otras muchas opciones.

- **Tortoise y Assembla**

Tortoise es un cliente para Windows del sistema de control de versiones Subversion, del cual ya se habló en la sección 2.6.1.
Para la creación del repositorio con el que se ha hecho el control de versiones se creó para el departamento de telefonía de la UC3M una cuenta en Assembla [19]. Assembla es una compañía que proporciona repositorios y seguimiento del proyecto de una aplicación en la nube. Para pequeños proyectos se pueden crear repositorios gratuitos para un grupo pequeño de desarrolladores. Para mayores características y usos más avanzados de la plataforma hay que pagar, pero para el desarrollo de esta aplicación no era necesario.
Se eligió utilizar Subversion y Tortoise porque la autora del proyecto ya había trabajado previamente con estas tecnologías por lo que suponía un ahorro de tiempo. Por otro lado, aunque Git sea más rápido y optimizado para la realización de *merge* (ver sección 2.6.1.), dado que sólo una persona trabajaba en el desarrollo de esta aplicación no era tan necesarias estas características.
La política de commits que se utilizó en la aplicación fue realizar al menos un commit al día con comentarios sobre el estado actual de la aplicación, además de commits adicionales una vez que se había terminado de implementar una funcionalidad y se había probado su correcto funcionamiento a través de baterías de pruebas manuales sobre el dispositivo móvil.

- **Web service de autenticación contra LDAP**

El protocolo LDAP permite el acceso y actualización de una base de datos especial llamada directorio. Estos directorios están optimizados para realizar búsquedas y lecturas

de datos de forma que son muy rápidos en ello. Para más información ver la sección 2.5.2. LDAP.
La Universidad Carlos III de Madrid utiliza este protocolo para su directorio de usuarios de la universidad. A la hora de la autenticación para el módulo de la consulta del consumo y tarifas telefónicas se necesita que el usuario se autentique contra dicho directorio LDAP de la universidad. Para ello, un desarrollador de aplicaciones web que trabaja en la universidad creó un web service contra el que se valida la aplicación en la pantalla de autenticación (login) de la aplicación.

- **Robotium**

Robotium es un framework de pruebas sobre Android, que proporciona una librería para facilitar la codificación de test funcionales, de sistema y de aceptación. Se ha utilizado en este proyecto la última versión disponible a día de hoy, la 4.3 [9]. Además, los desarrolladores de Robotium proveen acceso a la información de su API y varios tutoriales desde su Wiki [20].

3.5. Diagrama de casos de uso y diagrama y estructura de paquetes

- **Diagrama de casos de uso**

Un caso de uso representa el uso típico que da o puede dar un usuario a una aplicación. Muestra de una forma sencilla y resumida la funcionalidad que tendrá asociada cada tipo de usuario de la aplicación para facilitar dicha información al cliente.

En el caso de la aplicación de telefonía corporativa de la UC3M tan sólo hay dos tipos de usuario como se puede ver en el diagrama de casos de uso de la figura 16. El primer tipo, que se ha denominado “usuario normal” (en color rojo oscuro) se trata de cualquier miembro que trabaje en la universidad (PAS, PDI, empresas externas, Institutos, masters y doctorados) que tenga asociado una línea telefónica corporativa (fija o móvil). El segundo tipo de usuario (en color morado oscuro) representa a un responsable de grupo o área en la universidad. Este segundo tipo de usuario tendrá la funcionalidad añadida de la consulta de la facturación de la orgánica telefónica de la que es responsable.

[9] Robotium se puede descargar desde:https://code.google.com/p/robotium/downloads/list

En el diagrama de casos de uso de la Figura 17 se ha dado el mismo color de fondo a aquellas funcionalidades que se engloban dentro del mismo grupo funcional. Cada una de las funcionalidades y los grupos funcionales del diagrama serán explicados en la sección 3.6.

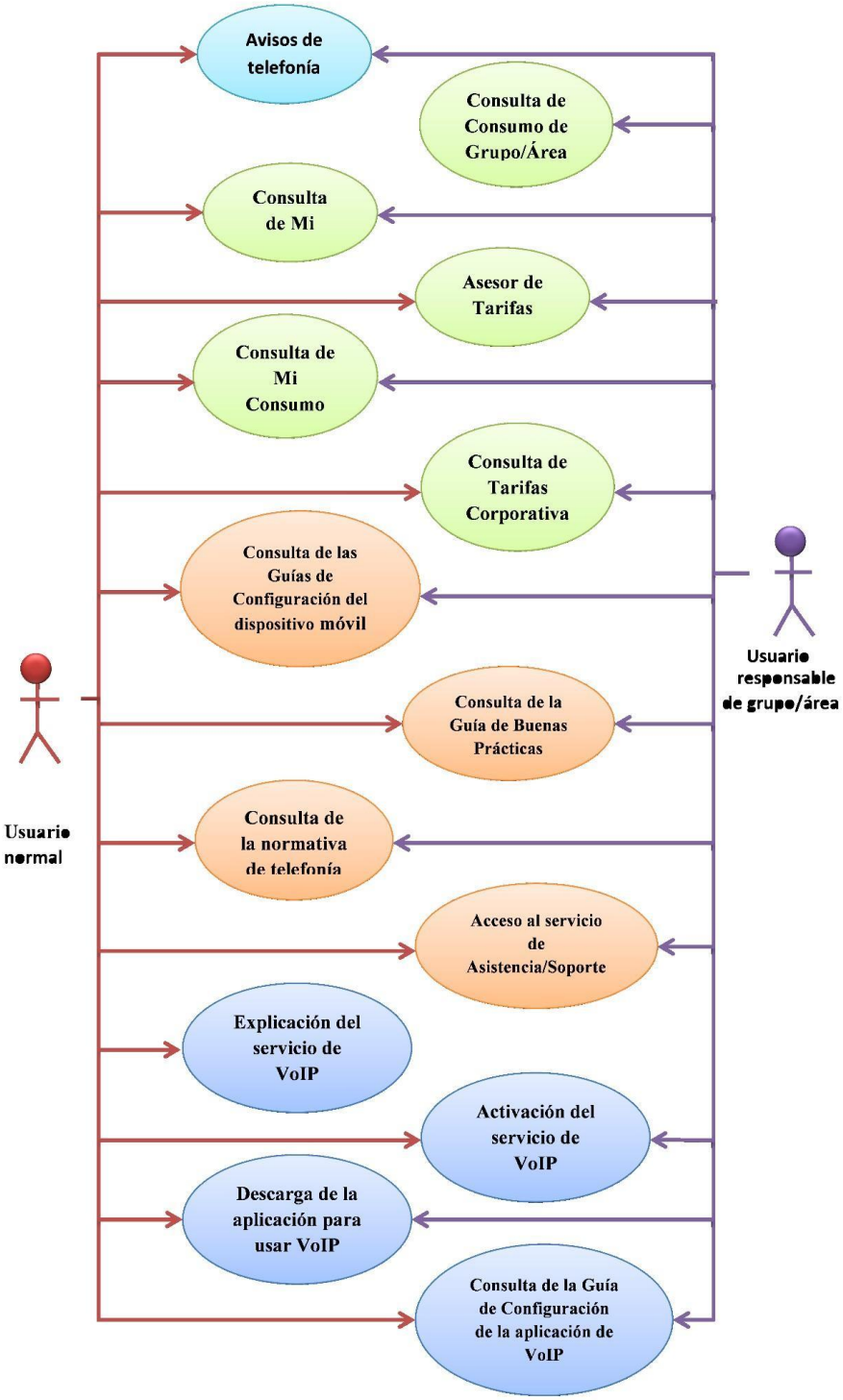

Figura 17 - Diagrama de casos de uso de la aplicación de telefonía corporativa de la UC3M

- **Estructura de paquetes**

En la Figura 18 se muestra el diagrama de paquetes en el que está estructurada la aplicación de telefonía corporativa de la UC3M. La aplicación está dividida en siete paquetes que engloban las clases con funcionalidades similares. Además, en la figura se puede observar la estructura de carpetas de recursos alternativos utilizados en la aplicación.

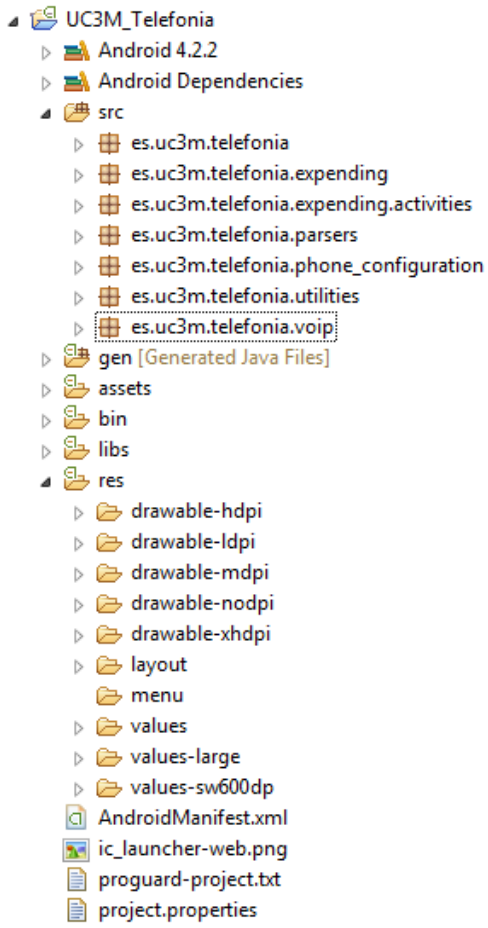

Figura 18 - Estructura de paquetes de la aplicación de telefonía corporativa de la UC3M

A continuación se va a explicar brevemente cada uno de los paquetes:

- es.uc3m.telefonia: Se trata del paquete que contiene la clase de la pantalla del menú principal y que sirve de estructura base para la ruta en la estructura de las clases para las clases autogeneradas como la R.java.

- es.uc3m.telefonia.expending: Este paquete contiene las doce clases que se utilizan para almacenar la información obtenida de los XML del área de "Facturación y Tarifas".

- es.uc3m.telefonia.expending.actividades: Este paquete contiene las catorce clases con las Activity y FragmentsActivity del área de “Facturación y Tarifas”.

- es.uc3m.telefonia.parsers: Este paquete contiene los siete parsers encargados de interpretar el archivo XML de respuesta de los web service de consumo y tarifas, de permisos y de autenticación de usuarios y almacenar dicha información en las diferentes clases creadas para ello que contiene el paquete de es.uc3m.telefonia.expending.

- es.uc3m.telefonia.phone_configuration: Este paquete contiene las siete Activity y FragmentActivity que conforman el área de “Configuración y Soporte”.

- es.uc3m.telefonia.utilities: Este paquete contiene las seis clases de utilidades que se utilizan en diversas partes de la aplicación tales como los Adapters personalizados, la clase de constantes, la clase con el cliente HTTP propio que usa sólo el esquema HTTPS, la clase que define la sesión del usuario una vez autenticado en el área de “Facturación y Tarifas” o la clase que implementa la funcionalidad del zoom en la imágenes de las guías de configuración.

- es.uc3m.telefonia.voip: Este paquete contiene las dos clases con las Activity del área de “Telefonía IP” de la aplicación.

- **Diagrama de paquetes**

En la Figura 19 se puede observar un diagrama de relación entre los paquetes de la aplicación. Cabe destacar el paquete de utilidades “es.uc3m.telefonia.utilities”, el cual es utilizado por todos los paquetes de la aplicación.

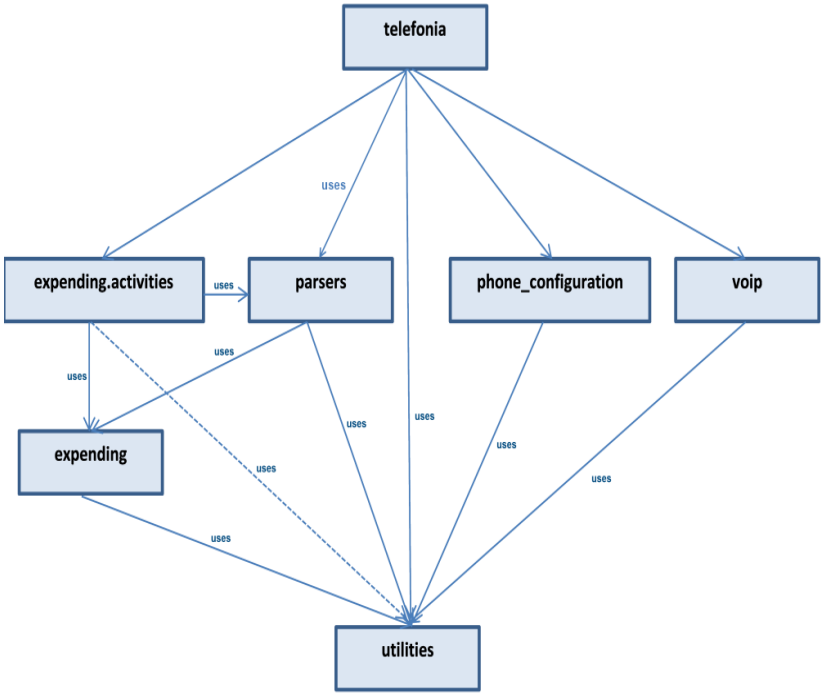

Figura 19 - Diagrama de paquetes de la aplicación de telefonía corporativa de la UC3M

3.6. Funcionamiento de la aplicación

La aplicación de telefonía corporativa de la UC3M se puede desglosar en tres grandes módulos funcionales. Cada uno de ellos cubre tres de los servicios más requeridos del departamento de telefonía: la consulta del consumo y las tarifas telefónicas, la configuración y soporte de smartphones y tablets corporativas y por último, la activación y configuración del servicio de VoIP de la universidad.

Además existe un cuarto módulo de funcionalidad sencillo, el de avisos del servicio de telefonía de la universidad.

Al iniciar la aplicación se le muestra al usuario una primera pantalla con una lista para acceder a los tres primeros módulos antes mencionados. Como hay varias listas en la aplicación para los menús que contienen además de texto, generalmente un icono y un color de fondo en las filas impares (cada fila empieza en el índice 0), se creó un *Adapter* personalizado del que se hablará más en detalle en la sección 3.7.6 Por ahora se dirá tan sólo que permite modificar los elementos de una lista fácilmente.

Por otro lado, se realiza la petición del texto de avisos a través de Internet. Como respuesta, se recibe un archivo XML con el texto que ha de mostrarse debajo de la lista del menú sobre los avisos. Dicho archivo XML se almacena en un repositorio de la universidad y suele contener avisos de la disponibilidad de la facturación de un nuevo mes o incidencias en el uso de la telefonía IP.
Si el dispositivo móvil no tiene conexión a Internet, se mostrará un texto apropiado informando al usuario. Se utiliza una barra de carga circular mientras se realiza dicha petición. Además la vista de los avisos tiene un botón para recargar los avisos.

En la Figura 20 se puede observar la pantalla de inicio de la aplicación en sus posibles estados: intentando descargar el texto de la imagen de avisos y descarga correcta del texto de avisos.

Figura 20 - Menú principal de la aplicación mientras carga la imagen de avisos (izq.) y menú principal con ejemplo de un aviso (dcha.)

Todas las pantallas de la aplicación están preparadas para su visionado tanto en vertical (portrait) como en horizontal (landscape) con sus desplazamientos de pantalla (scrolling) necesarios.

A continuación se van a explicar cada uno de los módulos de funcionalidad de la aplicación.

3.6.1. Facturación y Tarifas

Este módulo de funcionalidad permite al usuario acceder a diversos menús de consulta de la facturación y de las tarifas disponibles.

Actualmente la Universidad Carlos III de Madrid tiene un contrato con un operador de telefonía tanto para las líneas fijas como las líneas móviles. Este operador de telefonía envía un informe mensual del consumo de todas las líneas pertenecientes a miembros de la universidad al departamento de telefonía. Estas líneas se separan por su pertenencia al departamento que paga la facturación de dicha línea. Además se separan también en el consumo realizado por líneas fijas y líneas móviles. Sin embargo, este operador de telefonía tiene una peculiaridad en la creación de los listados de facturación y es que considera las llamadas a móviles, aunque sean realizadas desde un fijo, parte de la tarificación de líneas móviles. Esto puede llevar al principio a confusión a algunos usuarios ya que les aparece consumo de líneas "móviles" aunque no tengan ningún móvil corporativo. Sin embargo, como se especifica en todo momento el número de teléfono de cada consumo y el tipo de línea en el detalle del consumo es fácil ver que en realidad es gasto realizado con el teléfono fijo.

Hasta ahora, el único modo de que los usuarios de dichas líneas telefónicas pudieran conocer su consumo era poniendo una incidencia en Hidra (el gestor de Incidencias de la UC3M) al departamento de telefonía. Para facilitar este servicio la universidad ha contratado a una empresa externa para que realice y mantenga una página web en la que consultar los datos de consumo telefónico y estadísticas relacionadas con el consumo. Más adelante crearán web services (servicios web) con los datos reales del consumo y tarifas de los miembros de la universidad. La aplicación hará uso en un futuro de estos web services para la obtención de los datos de consumo y de tarifas, pero como aún no están creados, se simula este comportamiento obteniendo los archivos XML de respuesta del web service de unos archivos de prueba que se almacenan en el propio paquete de la aplicación.

Los métodos necesarios para la obtención de estos datos a través de los web service y sus posibles fallos en la obtención de la información (el dispositivo no está conectado a Internet o hay errores en el archivo de respuesta) están ya implementados y comentados en el código. Aunque no se han podido probar en un escenario real, al ser su funcionamiento muy similar al web service de autenticación de los usuarios no deberían fallar al poner la URL real de los web services que creen en la empresa que ha de realizar los web service de consumo y tarifas .

Los archivos XML de pruebas[10] creados para simular el archivo de respuesta de los web services se almacenan en la carpeta de `assets` del proyecto para que puedan ser

[10] Todos los nombres e información de consumo de los datos simulados son ficticios, pero basados en la estructura real de las tarificaciones que envía el operador de telefonía contratado por la UC3M.

accedidos desde el código. Estos archivos de prueba se han enviado a la empresa que ha de realizar los web service de facturación y tarifas como ejemplo real de la estructura que deben utilizar en la creación de los archivos XML de respuesta de sus web services. Dichas respuestas en XML contendrán la información de los usuarios que se requiera en la petición realizada por la aplicación de telefonía.

Para el acceso a todo este módulo de consulta se necesita autenticarse como usuario en la universidad. A día de hoy aún se pueden autenticar alumnos en la aplicación, pero se va a crear una nueva versión del web service de autenticación para que la rama del directorio LDAP de alumnos no sea incluida a la hora de validarse en esta aplicación.

En todos las opciones del módulo de consulta de consumo y de tarificaciones que necesitan acceder a internet para obtener los datos y manejar la respuesta en forma de archivo XML obtenida de los web service de consulta (es decir, todas menos el asesor de tarifas), se realiza el mismo control de los posibles errores en dicha acción. Se pueden dar distintos casos para los cuales se informará de forma distinta al usuario.

- No hay conexión a Internet: Se muestra al usuario un cuadro de diálogo informando de que hay un problema con la conexión a Internet y las posibles acciones a realizar: “Reintentar”, “Activar Internet” y “Cancelar”. La opción de “Reintentar” se utiliza en caso de que fuese un fallo puntual en la conexión a Internet (por ejemplo pasar por un túnel sin cobertura). La opción de “Activar Internet” se utiliza cuando el dispositivo no tiene habilitada ninguna conexión actualmente y se quiere activar o bien una conexión Wi-Fi bien una conexión de datos móviles. Esta opción lleva al usuario al menú de configuración de las redes en dispositivos con versión 2.3 y anteriores y a la pantalla de “Ajustes” en la versión 3.0 o posteriores ya que la red Wi-Fi pasa a activarse desde la pantalla principal de ajustes desde dicha versión. La última opción permite cerrar el cuadro de diálogo sin realizar ninguna acción.
  En la Figura 21 se puede ver el uso de este cuadro de diálogo.

Figura 21 - Cuadro de diálogo de error: error en la conexión a Internet

- Error en los datos: El error en los datos obtenidos se genera cuando hay algún error que no está contemplado en el sistema de manejo de los datos del archivo XML de respuesta. Se trata de un error que compromete la veracidad de los datos obtenidos. Se debe a errores en la construcción de los archivos XML del usuario por parte de la empresa encargada de crearlos, como por ejemplo no cerrar una etiqueta de los datos de una orgánica (`</organica>`) por lo que se mezclaría dicha orgánica con la siguiente dando lugar a datos erróneos. En este caso se le muestran al usuario de nuevo las opciones de "Reintentar" y "Cancelar" como en el caso anterior pero la opción de "Activar Internet" se cambia por la de "Llamar al CASO". Esta opción lleva a la pantalla de "Asistencia/Soporte" que permite llamar al CASO (Centro de Atención y Soporte de la UC3M) para poner la incidencia sobre el error en los datos.

El menú principal de este módulo se compone de las opciones de "Mi perfil", "Mi Consumo", "Consumo de Grupo/Área", "Asesor de Tarifas" y "Tarifas Corporativas" como se puede observar en la Figura 22:

Figura 22 - Menú de Facturación y Tarifas

En todos los menús, excepto en el asesor de tarifas, se presenta mucha cantidad de información de modo que se ha tratado de simplificar la presentación de esta información al usuario para no sobrecargar la pantalla. Para dichas pantallas se ha utilizado el sistema de pestañas, para separar la información de las líneas de telefonía fija y móvil principalmente. Además en algunas pantallas se hace uso del deslizamiento horizontal para moverse entre distinta información relacionada. Para el asesor de tarifas se diseñó un sistema de navegación especial, similar a las listas expandibles multinivel, que se explicará más adelante en esta misma sección.

A continuación se van a enumerar los bloques funcionales que componen el módulo de consulta de consumo y tarifas:

- **Autenticación de usuarios**

Para acceder a cualquiera de las opciones del menú de consulta de consumo y de tarifas el usuario ha de iniciar sesión con su cuenta de usuario de la universidad (la misma que para acceder a Campus Global, la intranet de la UC3M).
El usuario habrá de introducir su usuario (login) y contraseña en los campos de edición habilitados para ello. La aplicación envía por petición POST las credenciales al web service que valida dichos credenciales contra el LDAP, el directorio de los miembros de la universidad el cual se explicó en la sección 2.5.2. Una petición POST es aquella en la que no se manda en la propia URL la información a validar sino en el cuerpo de la petición. La

conexión con el web service se hace a través del protocolo https para una conexión segura y cifrada. Por último se le aplica un tiempo de expiración a la conexión para evitar posibles intentos de conexión infinitos por problemas en la conexión como por ejemplo, estar conectado a redes en las que se han de aceptar las condiciones de uso para tener una conectividad real a Internet.

Este web service de autenticación de usuarios sí está creado y en funcionamiento (no es emulado en local como el resto de web service de los que se tiene ejemplos de sus respuestas en la propia aplicación para simular su comportamiento).

El usuario ha de introducir un nombre de usuario (login) y la contraseña. En caso de faltar alguno de estos dos campos se mostrará un error informando del problema antes de realizar ninguna comprobación con el web service de autenticación (Figura 23).

Figura 23 - Pantalla de autenticación: error usuario y/o contraseña vacíos

Se realiza a continuación una verificación de si el usuario está conectado a Internet. En caso de no estarlo se le muestra un diálogo con las posibles acciones a realizar. Este diálogo y sus opciones es igual al visto antes en esta misma sección sobre problemas en la conexión a internet en la figura 20.

A la hora de iniciar sesión correctamente con un usuario se le permite guardar el nombre del usuario para futuros inicios de sesión y eliminar los nombres anteriormente guardados. Sólo se guardará el nombre de usuario si así se desea pero en ningún caso la contraseña. Si el usuario ha decidido guardar su nombre de inicio de sesión en futuros inicios aparecerá una lista con los posibles nombres con los que autocompletar al poner las

primeras iniciales de su nombre de usuario. Además si su usuario ya ha sido almacenado en la lista de usuarios guardados no se le volverá a preguntar al usuario si desea guardar el nombre (Figura 24).

Figura 24 - Pantalla de autenticación: posibilidad de guardar el usuario para futuros inicios de sesión (izq.) y un ejemplo de autocompletado (dcha.)

Una vez iniciada sesión, el usuario podrá salir a los demás módulos de la aplicación sin tener que volver a iniciar sesión cuando desee entrar de nuevo en el módulo de consulta de consumo y de tarifas. Se habrá de autenticar de nuevo en el caso de que la aplicación sea destruida por falta de memoria en el dispositivo móvil, o se pulse el botón de desconexión presente en las pantallas del área de “Facturación y Tarifas” y se confirme el abandonar la sesión.

- **Salir de la sesión**

Una vez iniciada la sesión aparecerá en el banner que contiene el título de la pantalla en la que está actualmente el usuario un botón para salir de la sesión. Este botón sólo aparece en las pantallas del menú de consulta de consumo y tarifas.
El usuario podrá salir de la sesión desde cualquiera de las pantallas del módulo de “Facturación y Tarifas”. Se le mostrará un cuadro de diálogo para que confirme que desea salir de la sesión y en caso de confirmar el cierre de la sesión se le redirigirá a la pantalla de inicio de sesión de nuevo. En la Figura 25 se puede observar el botón para salir de la sesión y el mensaje de confirmación:

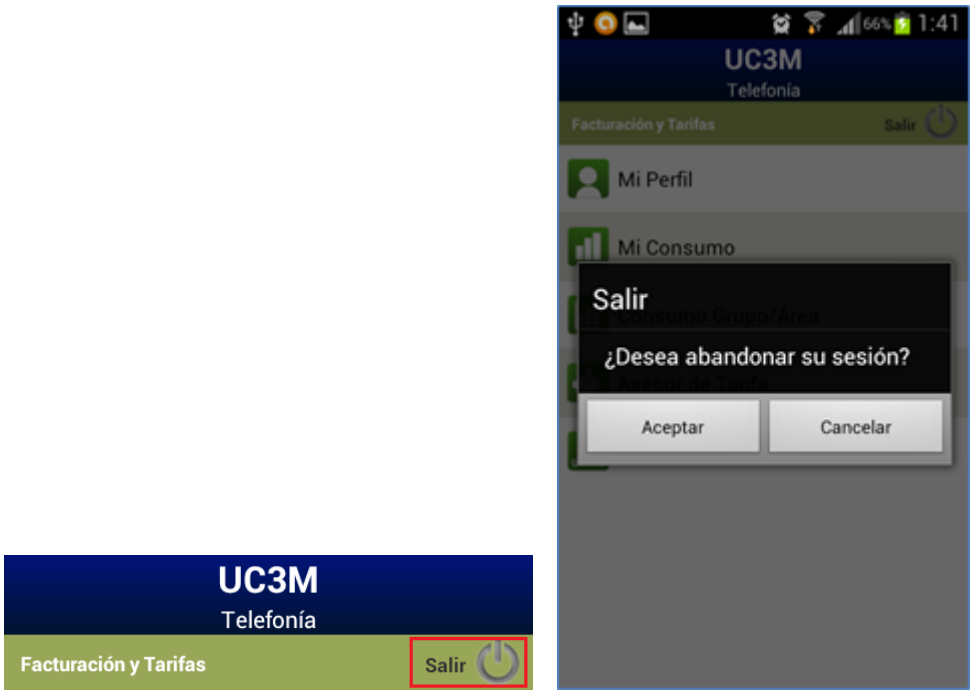

Figura 25 - Botón para salir de la sesión (izq.) y cuadro de diálogo para confirmar el salir de la sesión (dcha.)

Si el usuario vuelve para atrás con el botón *Atrás* (*Back*) del dispositivo móvil, la aplicación mostrará el menú principal (la pantalla previa a la de inicio de sesión) y no la pantalla en la que estuviera previamente a salirse de la sesión, ya que para acceder a cualquiera de ellas debe encontrarse autenticado en la aplicación.
La sesión se mantendrá abierta hasta que se cierre la sesión utilizando el botón habilitado para ello en la aplicación, o bien hasta que la aplicación sea eliminada por el sistema. Las aplicaciones mantienen su estado como caché por si se vuelve a inicializar en poco tiempo sea un inicio rápido, por lo que se mantiene hasta que el sistema las elimina por falta de memoria. Por lo que aunque hayamos salido de la aplicación yendo hacia atrás hasta la aplicación o el escritorio en el que se estuviera previamente a iniciar la aplicación, si esta se vuelve a abrir en un cierto tiempo, el usuario seguirá teniendo la sesión abierta y no tendrá que volver a autenticarse.

- **Comprobación de permisos del usuario**

La opción de consumo de Grupo/Área sólo será visible para los responsables de departamento. Antes de mostrar el menú principal de “Facturación y Tarifas” se realiza la comprobación de permisos del usuario pidiendo dicha información a otro web service, al que se le indica el nombre de usuario (login) y que devuelve un archivo XML en el que indica si el usuario tiene permiso para ver el menú de Grupo/Área.

- **Mi perfil**

En la opción de “Mi Perfil” (Figura 26) el usuario podrá acceder a información de los permisos de voz de cada una de sus líneas fijas y móviles (Ej. llamadas a nacional y móviles, llamadas internacionales...). En el caso de las líneas móviles también obtendrá información de sus permisos de datos (Ej. datos desactivados, datos nacionales, roaming...) para cada línea móvil y de la tarifa contratada por cada una de sus líneas móviles (Ej. tarifa plana 1GB, bono número VIP...).

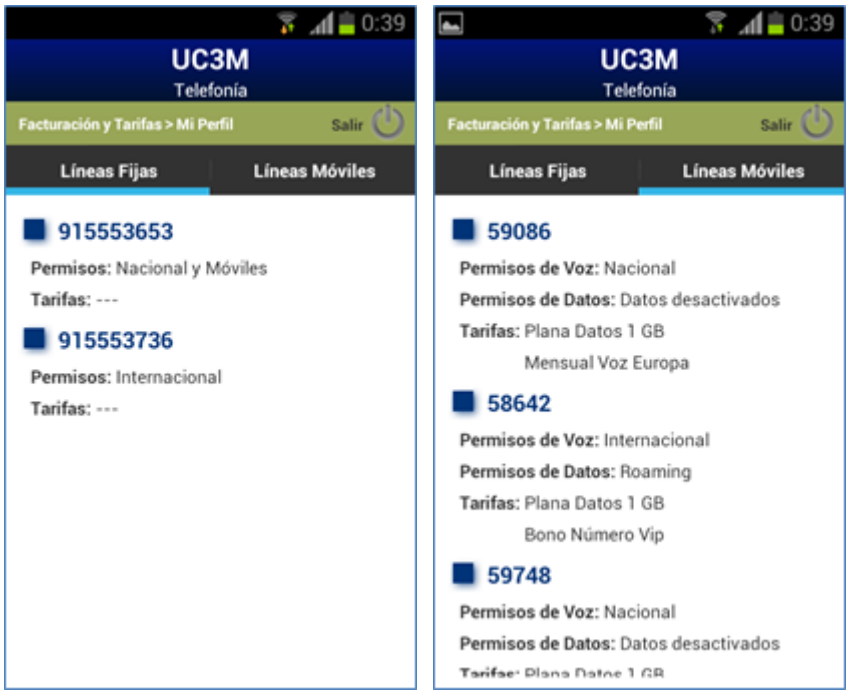

Figura 26 - Mi Perfil: Fijos (izq.) y móviles (dcha.)

- **Mi consumo**

En la opción de “Mi Consumo” (Figura 27) el usuario puede ver una tabla con sus diferentes líneas telefónicas fijas o móviles según la pestaña que se seleccione, con el consumo de cada una de ellas y el consumo total de todas las líneas fijas o móviles. Si selecciona una de las líneas telefónicas podrás ver el detalle del consumo de esa línea.

Figura 27 - Mi Consumo: Móviles

Esta información de consumo es la correspondiente a un periodo que tal y como realiza la facturación el operador de telefonía contratado por la universidad se corresponde con un mes del año normal. El usuario podrá elegir entre los periodos disponibles (Figura 28).

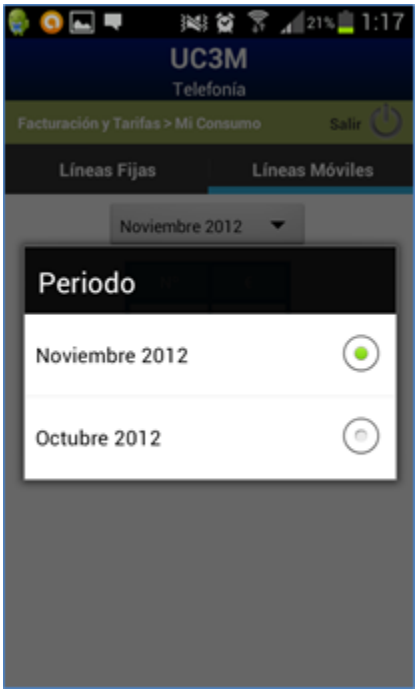

Figura 28 - Mi Consumo: Elección de periodo

El detalle de una línea (Figura 29) varía en función de si es una línea fija o móvil, y en el caso de las móviles en función del tipo de líneas (fijo que llama a móvil, móvil corporativo, modem, etc).

Figura 29 - Detalle de Mi Consumo: llamadas de fijos a móviles (izq.) y consumo de móvil corporativo (dcha.)

- **Consumo de grupo/área**

Esta opción sólo será visible para aquellos usuarios que sean responsables de departamento o área. Se muestran los datos de consumo de todas las líneas fijas y móviles asociadas a la orgánica del departamento. La información se muestra atendiendo a la elección de una orgánica (un usuario puede ser responsable de varias orgánicas de varios departamentos) y de un periodo como se puede observar en la Figura 30.

Figura 30 - Consumo Grupo/Área: Elección de orgánica y periodo

Una vez seleccionada una orgánica, por defecto se muestra la información del periodo más actual, pero se puede cambiar de periodo o de orgánica en cualquier momento. Esta primera tabla de información muestra por filas el número de línea telefónica, el nombre del usuario o lo usuarios asociados a dicha línea, y el coste mensual. Por último muestra el coste total telefónico de ese periodo para esa orgánica.

Esta tabla viene ordenada por defecto según el coste de cada línea para dicho periodo, pero también se puede ordenar por número de teléfono de menor a mayor, o por orden alfabético del nombre del usuario asociado a dicha línea telefónica. Para ordenarlos se pulsa sobre el título de cada columna. El modo de ordenación seleccionado se muestra con una flecha hacia arriba a la derecha del título de la columna, mientras que las otras dos columnas mantienen la flecha hacia abajo. Los datos de la tabla sólo se vuelven a cargar con la nueva ordenación si el modo es distinto al que estaba seleccionado en ese momento (Figura 31).

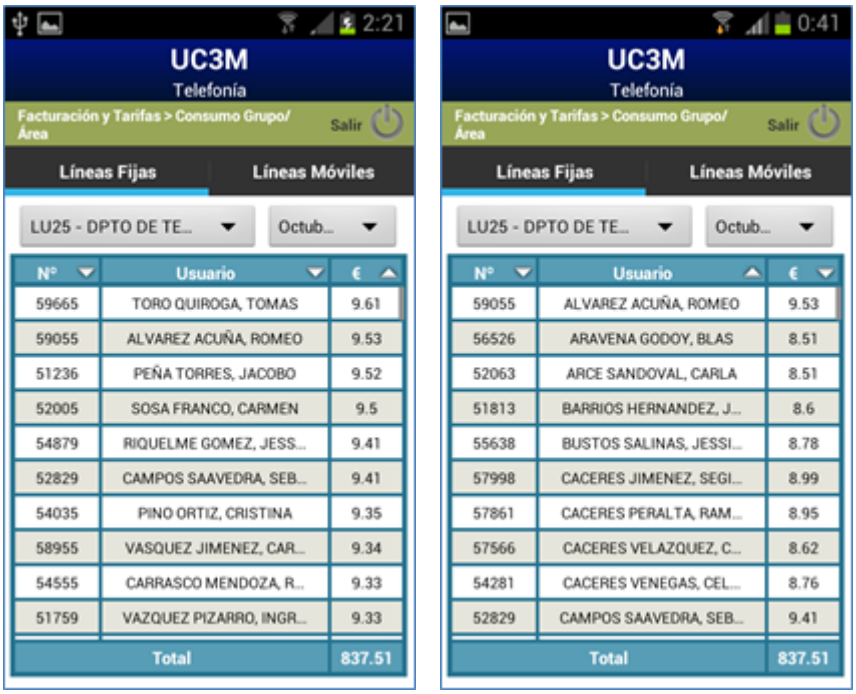

Figura 31 - Consumo Grupo/Área: Tabla de consumo con datos ordenados por precio (izq.) y datos ordenados por nombre del usuario (dcha.)

Al igual que en el apartado “Mi Consumo” se puede acceder a la información detallada de cada una de las líneas pulsando sobre cualquiera de las columnas de una misma fila perteneciente a una línea de teléfono (Figura 32).

Figura 32 - Consumo Grupo/Área: Detalle

- **Asesor de tarifa**

En la opción del “Asesor de Tarifa” se recomienda al usuario la tarifa más adecuada para contratar en su móvil corporativo a través de una serie de elecciones tales como “Utilizo el móvil en España” o “Utilizo el móvil fuera de España” o bien “Llamo a destinos Europeos” o “Llamo a destinos fuera de Europa”.
Las opciones que se van seleccionando se siguen mostrando en la pantalla y las opciones no elegidas se eliminan de la pantalla, por lo que se puede ver todo el flujo de opciones elegidas y regresar a algunas de las elecciones anteriores pulsando sobre la opción que se eligió.

Usaremos un ejemplo para ilustrar su funcionamiento: el asesor de tarifa da a elegir al usuario en primer lugar si quiere tener una tarifa plana o pagar por consumo (Figura 33 y Figura 34). Si decide pagar por consumo llevará al usuario a la pantalla de información de todas las tarifas de consumo del menú “Tarifas Corporativas”.

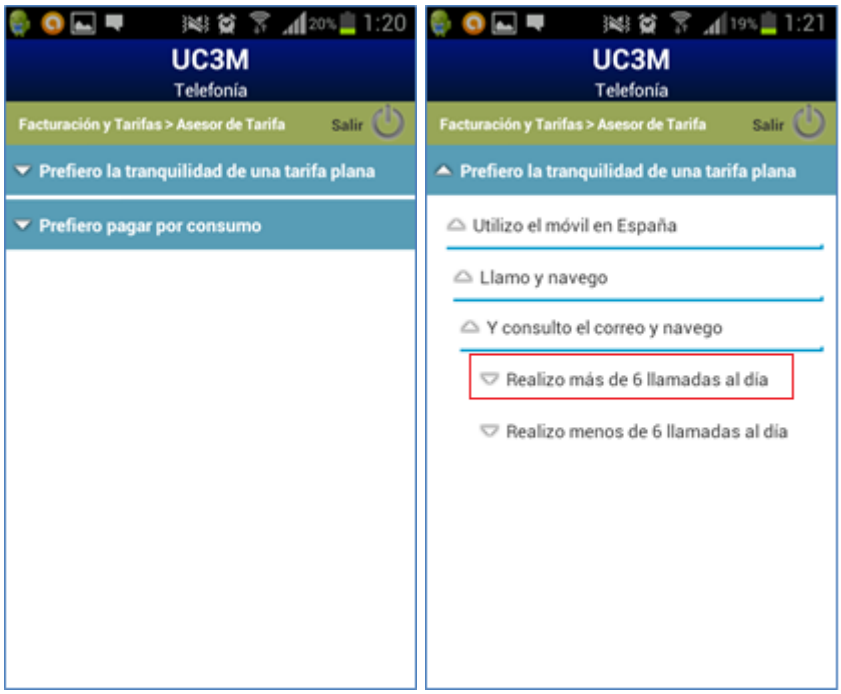

Figura 33 - Asesor de Tarifa: Primera elección (izq.) y ejemplo de seleccionar una opción (dcha.)

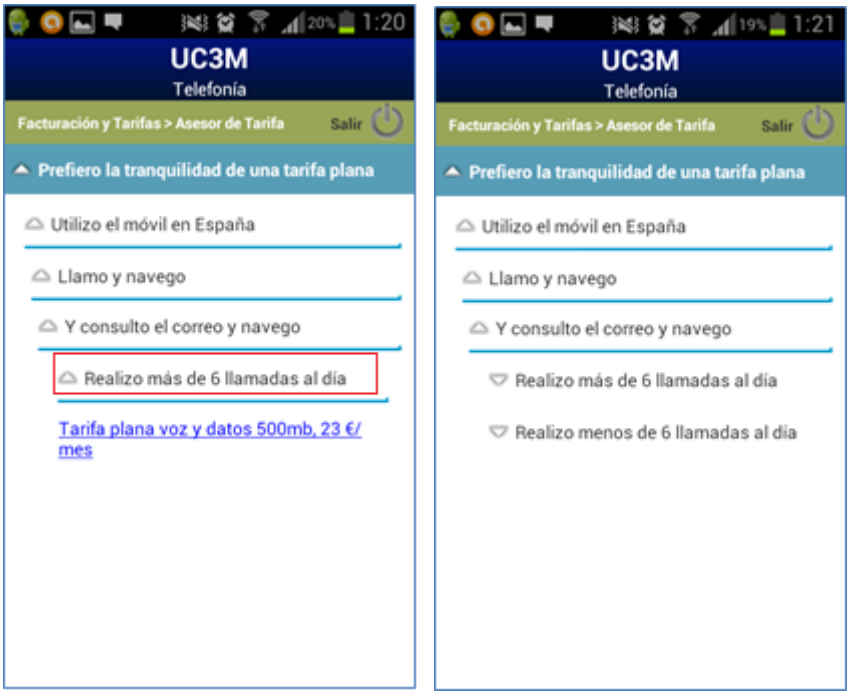

Figura 34 - Asesor de Tarifa: Ejemplo de mostrar la tarifa recomendada tras la última elección y querer mostrar de nuevo la anterior elección (izq.) y ejemplo de mostrar de nuevo la anterior elección (dcha.)

- **Tarifas corporativas**

En esta opción se muestra información de todas las tarifas corporativas que ofrece el operador contratado por la universidad a los miembros de la universidad, dividida por tarifas de consumo de voz, tarifas planas y tarifas de roaming y haciendo uso de la navegación en pestañas (Figura 35). A su vez cada una de estas está dividida en fijo y móvil excepto roaming que sólo es para móviles (Figura 36), haciendo uso del desplazamiento horizontal como método de navegación entre las tarifas para fijos y móviles.

En estas tablas se encuentra información sobre las tarifas más utilizadas pero se puede acceder a todas a través del enlace que se encuentra debajo de cada una de las tablas de tarifas.

En la Figura 35 y la Figura 36 se pueden ver algunos ejemplos de pantallas de las tarifas corporativas.

Figura 35 - Tarifas Corporativas: Consumo de voz para teléfonos fijos (izq.) y tarifas planas para teléfonos móviles (dcha.)

Figura 36 - Tarifas Corporativas: Tarifas de roaming para teléfonos móviles

3.6.2. Configuración y Soporte

En este módulo de funcionalidad se encuentra información para ayudar al usuario a configurar su smartphone o tablet corporativo y más información importante para el uso de los terminales móviles corporativos. Cuando para su trabajo en la universidad el usuario precisa de un teléfono móvil, en función de su necesidad (sólo llamar y poder ser localizado, conexión siempre a Internet, etc.) se le prestará desde el servicio de informática y comunicaciones (SdIC) un terminal adecuado. A estos terminales se les denominan teléfonos corporativos. Además según las necesidades del usuario tendrá activos distintos servicios de voz (llamadas nacionales, internacionales, etc.) y tendrá activada una tarifa de datos o no. Todos los gastos asociados a dichos servicios correrán a cargo de la orgánica del departamento al que esté asociada la línea.

Desde el departamento de telefonía sólo se brinda soporte a los teléfonos entregados y por lo tanto homologados de la universidad. A día 31 de Mayo de 2013 los dispositivos móviles homologados por la universidad con sistema operativo Android son el HTC Desire HD, el Samsung Galaxy SII, la Samsung Galaxy Tab y la Samsung Galaxy Tab 10.1".

En la Figura 37 se puede observar el menú principal de este bloque funcional.

Figura 37 - Configuración y soporte: menú principal

A continuación se habla sobre cada una de las opciones de este bloque funcional.

■ Configuración del smartphone

En la “Configuración del Smartphone” se presenta un resumen de las guías de configuración creadas por el departamento de telefonía para los dispositivos homologados por la universidad (Figura 38). En este resumen se han incluido las guías que desde el departamento de telefonía se han considerado más importantes para la configuración de teléfono corporativo: Introducir SIM, activar el teléfono, configurar Eduroam (servicio mundial de movilidad segura desarrollado para la comunidad académica y de investigación que provee de una red Wi-Fi segura habilitada en las instituciones participantes), configurar el correo electrónico corporativo, configurar la VPN (Virtual Private Network o red privada virtual) y como activar/desactivar los datos móviles.

Figura 38 - Configuración del Smartphone: menú de guías de configuración

Todas estas guías de configuración están diseñadas para mostrar al usuario lo que ha de hacer de forma rápida y clara. Para ello, están formadas por secuencias de imágenes del propio terminal señalando con recuadros rojos las opciones que debe pulsar el usuario y ejemplos de los datos con los que se ha de rellenar los formularios. Además estas imágenes son ampliables utilizando la técnica de pinch to zoom pinchando la imagen con dos dedos y separándolos para acercar la imagen de forma gradual y juntándolos para alejar la imagen de nuevo. También se puede pulsar dos veces sobre la imagen rápidamente (doble tap) para acercar al máximo y de nuevo dos pulsaciones rápidas para volver al zoom normal. Para pasar entre imágenes se desliza la pantalla hacia la izquierda para pasar a la siguiente imagen y se desliza hacia la izquierda para acceder de nuevo a la imagen anterior. Cada imagen está titulada con la palabra “Paso” más el número de imagen en la guía. En los títulos de las imágenes se puede observar si hay más imágenes

a continuación o no, porque aparece el título de la siguiente imagen existente en gris claro.

Por defecto las imágenes que se muestran son las del obtenidas en un Samsung Galaxy SII en caso de que la aplicación se esté ejecutando en un handset, y las de la Samsung Galaxy Tab 10.1" en caso de que se ejecute en una tablet.
Entre handset los menús no cambian mucho. Sin embargo, si alguien está utilizando la aplicación en una tablet, espera encontrar las imágenes de la guía de configuración de la tablet que tiene entre sus manos, o al menos de una tablet también, por lo que se decidió incluir imágenes de las guías de configuración específicamente para tablets también.
Por último, en la guía de configuración de insertar la SIM, además identifica si es el HTC Desire HD en el que se está ejecutando y muestra su guía de configuración para la introducción de la SIM. Se decidió no crear la guía específica del terminal HTC Desire HD ya que queda poca gente en la universidad con este terminal.

A continuación se muestra un ejemplo de la guía de configuración de la red Wi-Fi Eduroam para un teléfono tipo handset (Figura 39).

Figura 39 - Configuración del Smartphone: Configurar Eduroam

- **Normativa de telefonía**

En este menú se presenta de forma resumida la normativa para el servicio de telefonía de la universidad. Aún resumida, se trata de una gran cantidad de información a mostrar al usuario. Se estuvo pensando la mejor manera de hacerlo, y finalmente se decidió optar

por una adaptación del patrón de interacción de navegación en Android *Data drill down* [21].

Este patrón indica que se ha de mostrar una lista de opciones al usuario, cuando éste soluciona alguna se le pasa al usuario a la siguiente pantalla, también llamado el siguiente nivel. Se realiza una pequeña animación de deslizamiento horizontal de derecha a izquierda de la pantalla para crear un efecto de inmersión dentro de esa opción, es decir, la pantalla actual parece que desaparece por el lado izquierdo de la pantalla del dispositivo y la nueva pantalla del nuevo nivel aparece desde la derecha de la pantalla del dispositivo. Se puede volver atrás pulsando en el botón “Back” del dispositivo y se creará una animación nuevamente, pero esta vez de forma inversa, de izquierda a derecha.

Como se puede llegar hasta un nivel muy profundo de pantallas ya que es mucha cantidad de información, también se ha creado un botón en la parte superior de la pantalla para volver directamente a la lista inicial de contenidos de la normativa. Desde esta pantalla inicial también se podrá descargar la normativa completa desde un enlace en la parte inferior de la pantalla.

En la Figura 40 se puede ver un ejemplo de algunas de las pantallas de la normativa de telefonía a distintos niveles de profundidad:

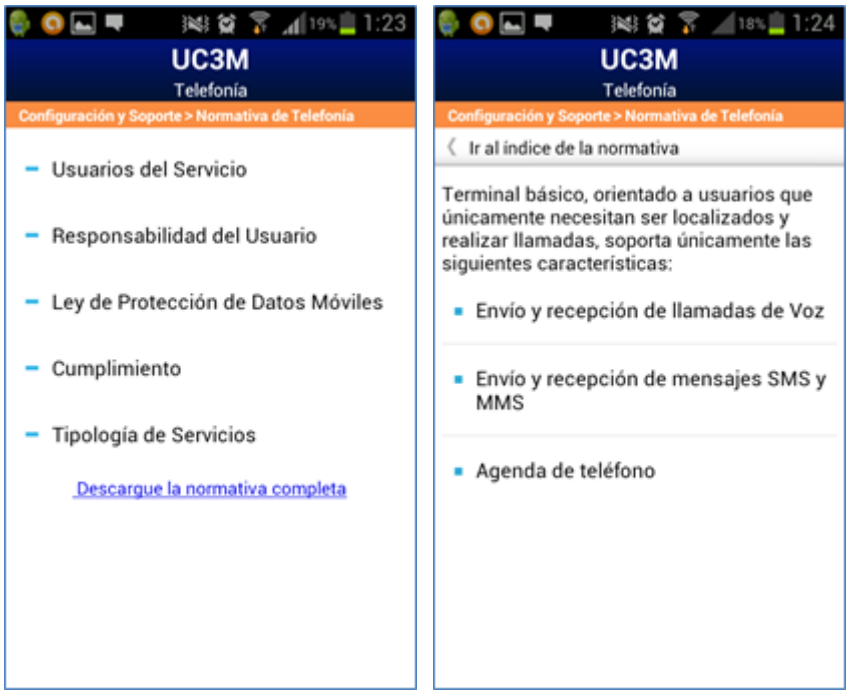

Figura 40 - Normativa de telefonía: Índice de contenidos (izq.) y ejemplo de una pantalla de la normativa (dcha.)

- **Guía de buenas prácticas**

En esta sección se dan consejos durante la utilización de los servicios de telefonía de la universidad. Por ejemplo, en qué casos mejor utilizar el teléfono fijo corporativo o el móvil corporativo para un mayor ahorro en consumo de voz, o para ahorrar en el consumo de datos y cuando sale del país con el teléfono corporativo (Figura 41).

Desde cualquiera de estas pantallas de información puede acceder a la guía completa de buenas prácticas pulsando sobre el enlace creado para tal fin.

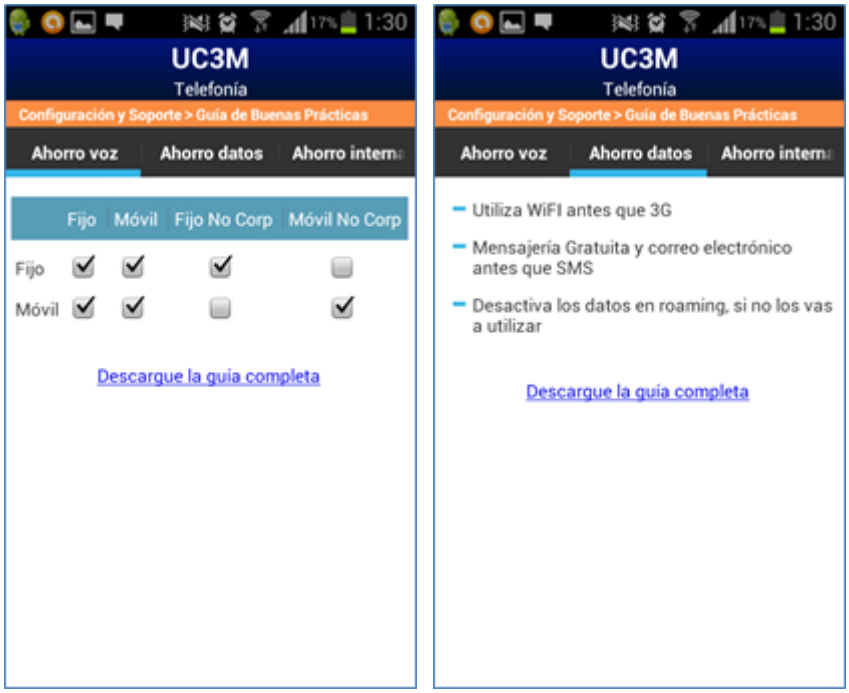

Figura 41 - Guía buenas prácticas: Ahorro voz (izq.) y ahorro datos (dcha.)

- **Asistencia/Soporte**

Este menú permite al usuario abrir la aplicación de marcación telefónica de su teléfono para llamar al CASO, el Centro de Atención y Soporte de la Universidad Carlos III de Madrid para poner una incidencia (Figura 42). Si la SIM del teléfono desde el que se ejecuta la aplicación es un número de teléfono corporativo se puede seleccionar que marque en la aplicación del marcador telefónico directamente la extensión para llamar al CASO, el 6200. En caso de que no sea un número corporativo se seleccionará que se marque el número largo del CASO, el 916246200. En ningún caso se llama directamente, sólo se le muestra el número de teléfono a llamar al usuario y será decisión de éste el realizar finalmente la llamada o no.

Figura 42 - Asistencia/Soporte: Opciones para llamar al CASO (izq.) y ejemplo para llamar desde un móvil corporativo (dcha.)

3.6.3. Telefonía IP

El servicio de telefonía IP de la universidad (VoIP) permite realizar llamadas desde fuera de la universidad como si se estuviera utilizando el teléfono fijo de la universidad con las mismas tarifas y permisos. Sobre todo es útil cuando el usuario ha de desplazarse fuera de España y necesita llamar a España, pues la tarifa que se le aplica es como si estuviera llamando desde la universidad.

En este módulo funcional se puede realizar lo necesario para contar con este servicio en un dispositivo móvil Android (Figura 43).

Figura 43 - Telefonía IP: menú principal de VoIP

- **Activación del servicio de VoIP**

Todo miembro de la universidad que disponga de una línea de telefonía fija puede activar el servicio de telefonía IP. Cuando se selecciona esta opción de la lista del menú de VoIP se abre en el dispositivo el navegador por defecto (o deja elegir al usuario que navegador quiere abrir), mostrando la aplicación web del servicio de telefonía IP (Figura 44). Para activarlo se ha de entrar con el usuario y contraseña de Campus Global y activar el servicio. Dado que el número de telefonía IP que le proporciona automáticamente la aplicación web se crea basándose en los datos del directorio de la universidad del usuario, el usuario debe tener estos datos actualizados.

Figura 44 - Telefonía IP: Activación del servicio de VoIP

- **Descarga de la aplicación de VoIP**

Esta opción muestra en el market por defecto del dispositivo móvil, generalmente Google Play, la aplicación Zoiper para que el usuario pueda descargarla e instalarla en su terminal (Figura 45). La aplicación Zoiper permite configurar cuentas de telefonía IP para realizar llamadas de VoIP. Es la que se recomienda ahora mismo desde el área de telefonía de la universidad debido a su calidad de sonido y volumen en las llamadas, a su fácil configuración, el uso de la misma interfaz en la aplicación móvil que de ordenador y a que es gratuita.

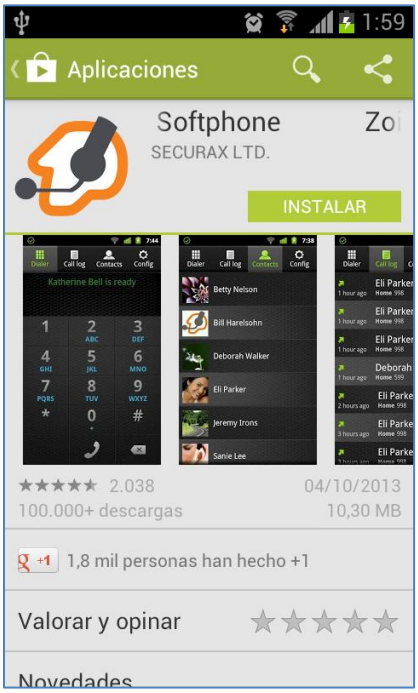

Figura 45 - Telefonía IP: Descarga de la aplicación Zoiper

- **Manual de configuración**

Por último, en la opción de manual de configuración se encuentra otra guía de configuración muy similar a las guías de configuración del módulo de “Configuración y Soporte” para la configuración de la aplicación Zoiper de telefonía IP con un ejemplo ilustrativo de cómo ha de configurar un usuario su cuenta (Figura 46).

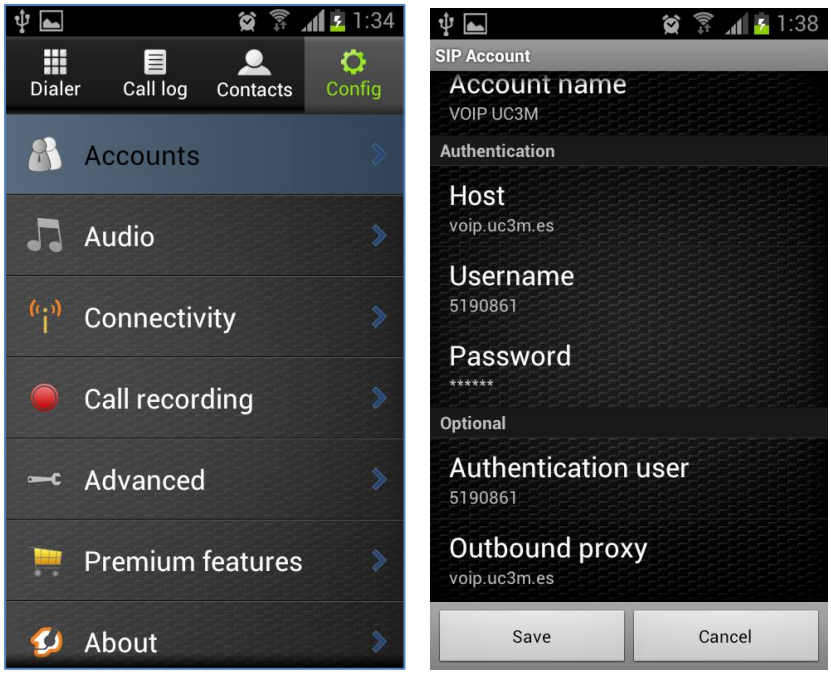

Figura 46 - Telefonía IP: Manual configuración paso 2 (izq.) y paso 4 (dcha.)

3.7. Características del sistema Android utilizadas en el desarrollo de la aplicación

A continuación se expondrán las características Android más reseñables utilizadas y una breve explicación sobre su uso en la aplicación. El uso genérico de la mayoría de estas características ya fue explicado en la sección 2.3 por lo que la explicación se centrará más en el uso de dichas características en la aplicación. Si desea el lector más información general sobre dicha característica puede consultar la sección 2.3 de la presente memoria.

Para mayor organización se han agrupado por grupos las características.

3.7.1. Uso del AndroidManifest

- **Dar permisos a la aplicación**

La aplicación necesita de los siguientes permisos:

- Comprobar el estado de la red: Se necesita este permiso para comprobar si el usuario tiene una conexión activa a Internet para el inicio de sesión en la sección de “Facturación y Tarifas”.

- Internet ilimitado: para la obtención de los datos del consumo y el aviso de la pantalla de avisos.

Estos permisos se muestran al usuario a la hora de instalar la aplicación y decide si aceptar aun así instalar la aplicación o no.

- **Aplicar temas**

Un tema (*theme*) es un estilo (*style*) como los explicados en la sección 2.3.3 pero en lugar de aplicarse a un objeto `View` se aplica a toda la aplicación por defecto. En el fichero `AndroidManifest` se puede definir el tema que llevará la aplicación como un atributo de la etiqueta `<application>`. Este tema puede ser uno de los existentes en Android como el `@android:style/Theme.Light.NoTitleBar` que dota a la aplicación de una interfaz de usuario clara (pantalla en blanco principalmente y letras oscuras) y sin la barra del título de la aplicación, o el tema `android:Theme.Black` que dota a la aplicación de una interfaz oscura (pantalla en negro principalmente y letras claras) y con la barra del título de la aplicación. Además de los múltiples temas aplicables disponibles en Android, se pueden definir temas propios en el fichero de estilos, creándolos bien desde cero o bien partiendo de alguno de los existentes (tanto de Android como de otros creados por el desarrollador).

En el caso de esta aplicación se ha creado un tema propio para la aplicación utilizando como tema base el `Theme.Light.NoTitleBar` que proveen las librerías de Android, es decir, fondo blanco con texto oscuro. Para mantener la consistencia entre versiones se decidió no utilizar el tema Holo en aquellos terminales que pudieran utilizar dicho tema para mantener el mismo tipo de botones, spinners, y demás elementos gráficos que varían notablemente.

- **Guardar en la tarjeta de memoria externa SD la aplicación**

Aunque la aplicación ocupa algo menos de 13 MB, se ha considerado que era importante dejar al usuario que pudiera almacenar la aplicación en la tarjeta de memoria externa SD. Hay que tener en cuenta si la aplicación necesita estar siempre activa como es el caso de aplicaciones de mensajería tipo Hangouts o Whatsapp o bien si no es una problemática mayor el que no funcione si se extrae la tarjeta SD o se desmonta la tarjeta SD como ocurre cuando se conecta el dispositivo móvil al ordenador para intercambio de datos.
Para permitir al usuario esta opción se define el atributo `android:installLocation` de la etiqueta `<manifest>` a "auto". Otros valores son “preferExternal” e “internalOnly”. Este último es el que se utiliza por defecto si no se le define uno en concreto para la aplicación en el fichero `AndroidManifest`.

- **Definir el icono de la aplicación**

Se ha definido un icono propio para mostrar en la lista de las aplicaciones disponibles en el dispositivo móvil. Dicho icono se guarda en las carpetas de recursos alternativos de `drawable` como las otras imágenes con sus distintos tamaños, y se referencia en el atributo `android:icon` dentro de la etiqueta `<application>` del fichero `AndroidManifest`.

3.7.2. Recursos de la aplicación

- **Externalizar recursos**

Se han definido todo lo posible en ficheros de recursos XML de la aplicación los recursos tales como cadena de caracteres (strings), estilos (styles), gráficos (drawables) tanto en archivos de imagen PNG como algunos creado por código en ficheros XML), arrays, colors, layouts, etc De este modo se separa el aspecto más gráfico como pueden ser los layouts o los colores de la funcionalidad propiamente dicha, que se le otorga por código. E incluso permitiría trabajar por separado al diseñador gráfico.
Sin embargo, en algunas ocasiones como en el consumo de grupo, el número de filas en la tabla es variable y dinámico por lo que se crean layouts específicos para crear cada una de las filas y de las celdas de la tabla.

- **Utilización de recursos alternativos**

Los recursos alternativos es una opción que provee Android para obtener los recursos más adecuados a cada terminal en tiempo de ejecución de aquellos disponibles en la aplicación. Por ejemplo, utilizar un archivo de strings (cadenas de caracteres) en inglés si nuestro dispositivo está en inglés, o español si está en español, utilizar distintos tamaños de imagen en función de si la pantalla es grande o si es pequeña, etc. Se han hablado en profundidad de ellos en la sección 2.3.3.

En esta aplicación se han utilizado gráficos (drawables) de distintos tamaños y estilos para distintas densidades de pantallas: mdpi, xhdpi, etc. de los iconos de los menús o de las imágenes de las guías de configuración.
También se han utilizado layouts o estilos distintos según si el tamaño de la pantalla es normal o x-large (para tablets) creando versiones de los layouts específicas para tablets.
Por último sería sencillo traducir la aplicación a otros idiomas pues todos los textos se encuentran en ficheros externos XML. Sin embargo para una traducción completa la empresa encargada de los web service de consumo debería traducir los textos de la facturación y las tarifas. Esto es debido a que los nombres y precios de las tarifas varían relativamente fácilmente, por lo que se ha intentado que la aplicación no dependa de que

el usuario actualice la aplicación con los nuevos nombres de las tarifas y sus costes evitando también la molestia de tener que actualizar a menudo la aplicación por cambios nimios como esos. Para ello, los nombres de tarifas y otros textos relacionados con la facturación se incluyen en los archivos XML de respuesta de los web services como otra dato más, y se han construido la estructura de datos y los parseadores de forma que puedan lidiar con cambios en los nombres de tarifas y en la cantidad de tarifas y de datos que se analizan en cada facturación.
Aunque actualmente no esté contemplado que la aplicación se traduzca a más idiomas, los pasos a seguir serían: primero, traducir todos los textos e incluirlos en las carpetas de recursos externos, por ejemplo `values-en` para inglés, y segundo, a la hora de enviar la petición al web service de consumo y tarifas para la obtención de los datos se incluiría el parámetro del idioma en el que se requiere la información.

- **Crear recursos propios**

Se han creado recursos propios como los iconos de los menús en formato PNG incluidos directamente en las carpetas de drawables alternativas según su tamaño.
Además se han creado otra serie de recursos directamente en código XML: selectores de imágenes en función del estado de la View (presionado, con el foco, etc.); el fondo del banner del título haciendo uso de un gradiente; las celdas de las tablas ya que las tablas en Android por defecto no tienen bordes (ni internos ni externos) las tablas; etc. Tanto el gradiente para el banner como las celdas de las tablas se han realizado utilizando la etiqueta *shape*, que permite crear formas básicas (rectángulos, óvalos, líneas o anillos) y darles un color de relleno, de borde, el tamaño del borde, esquinas redondeadas, etc. entre otras características.

3.7.3. Relativo a Activities

- **Uso de la pila de actividades y tareas**

Las actividades en Android son las encargadas de interactuar directamente con el usuario. Son por lo tanto las encargadas de mostrar el contenido en la pantalla y manejar la interacción del usuario con los distintos elementos presentes en esa pantalla, como por ejemplo la pantalla con los contactos del usuario en el teléfono. Para información más detallada consultar la sección 2.3.7. Android maneja el paso entre estas actividades con el concepto de *pila*. De este modo cuando por ejemplo al pulsar un botón te lleva a una nueva actividad (es decir, una nueva pantalla), la nueva actividad se coloca al principio de la pila de actividades y la anterior actividad en las que se encontraba el usuario para a la segunda posición, y así sucesivamente. Esto permite llevar un control del flujo del usuario a través de las pantallas y que al pulsar el botón de volver atrás del teléfono vuelva a la anterior actividad.

Este cómodo comportamiento suponía un problema en la zona de consumo de la aplicación al salirse el usuario de su sesión, ya que con el comportamiento por defecto, al darle al botón de atrás volvería al contenido de consumo sin haber iniciado de nuevo la sesión, dando lugar a un problema de seguridad.
Para resolverlo en un primer momento se implementó una comprobación en cada una de las actividades del consumo en el método `onResume()` del ciclo de vida de la actividad, ya que a menos que haya habido muy poca memoria y el sistema operativo haya decidido eliminarla, ésta simplemente se relanzará, por lo que siempre pasa por el método `onResume()`. Se puede consultar información sobre el ciclo de vida de una actividad en la sección 2.3.6. Si ningún usuario ha iniciado sesión (o acaba de cerrarla y por lo tanto ya no tiene ninguna sesión abierta) se llama al método `finish()` de la actividad para cerrarla y eliminarla de la pila de actividades por lo que se eliminan todas las actividades hasta volver a estar en la actividad del menú principal.
Aunque este sistema funciona bien para el caso de que el usuario salga de su sesión e intente volver atrás con el botón *Back*, existía un caso para lo cual esto no era válido: tras salir de la sesión si volvía a iniciarla inmediatamente, ya que tras salir de la sesión se devuelve al usuario a la pantalla de inicio de sesión. En caso de que fuera el mismo usuario el mayor problema es que tendría un comportamiento extraño no ir al menú de consumo sino a la anterior pantalla desde la que se salió de su sesión. El problema es que si se iniciaba sesión con otro usuario, y daba al botón de volver atrás, pasaba el control de seguridad antes mencionado porque había alguien autenticado. Sin embargo, como la actividad no se ha recreado, porque estaba en la pila de actividades, sólo se ha relanzado, los datos se corresponden con los del anterior usuario con lo que daba lugar de nuevo a un bug de seguridad.

Para resolver definitivamente este problema se encontró el uso de *flags*. Para llamar a una nueva actividad se hace uso de intents. En un intent se puede especificar bien explícitamente qué actividad quieres lanzar (por ejemplo si quieres lanzar otra actividad de tu propia aplicación) o bien implícitamente en la que estableces que por ejemplo quieres abrir un navegador y el sistema operativo muestra en un cuadro de diálogo los distintos navegadores que tiene en su dispositivo el usuario. Estos intents pueden llevar unos flags asociados a la hora de lanzar la nueva actividad, los cuales permiten modificar el comportamiento por defecto de un intent a la hora de lanzarse la actividad. Por ejemplo, cuando a un intent se le establece el flag `Intent.FLAG_ACTIVITY_NO_HISTORY` a través del método `setFlags()` implica que la nueva actividad no se guarda en la pila de actividades de la tarea. En cuanto el usuario navega fuera de dicha actividad, la actividad es destruida. Este flag se suele utilizar por ejemplo para las imágenes de inicio de una aplicación, a la que no ha de volverse en ningún momento.

Los flags utilizados para solucionar el problema del login fueron `Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK`. Estos flags se asocian al intent que lanza la actividad del menú de “Facturación y Tarifas”. Lo que hace es crear una nueva tarea y eliminar las actividades que se lanzaron anteriormente hasta la raíz de esta nueva tarea, es decir, busca si hay una instancia de la

clase que se quiere lanzar (en este caso el menú de consumo) y elimina todas las actividades de esa tarea hasta llegar a la actividad del menú de consumo y recrea ésta pasándole este nuevo intent.
Por ejemplo, si se tienen en la pila las siguientes actividades A - B - C - D - E, en E se sale de su sesión el usuario, por lo que se vuelve a ir a la actividad B de inicio de sesión; después, inicia sesión con éxito y pasa a la actividad C y la actividad B se elimina de la pila de actividades. Al iniciar la actividad C se le asocian estos flags por lo que las actividades D y E se eliminan de la pila y C se vuelve a recrear.
Por lo tanto la actividad es como si fuera creada por primera vez y vuelve a descargar todos los datos del usuario. Al presionar sobre el botón de volver atrás volverá al menú principal de la aplicación, no a la actividad E.

- **Uso del ciclo de vida de una actividad/fragment**

En algunas de las actividades de la aplicación es necesario guardar el estado en el que se encontraba en caso de que el usuario salga de la actividad o gire el dispositivo móvil (ya que al girarlo el comportamiento por defecto de Android es recrear la actividad).
Este es el caso de la pantalla de consumo de grupo / área en la que se ha de guardar el estado del spinner del periodo explícitamente. En caso de no guardarse el estado explícitamente, al volver a iniciar la aplicación, el spinner de la orgánica sí se queda guardado automáticamente el estado en el que estuviera (mostrando la orgánica que estuviera mostrando anteriormente). Sin embargo, el valor del periodo se resetea al primer valor por defecto, en lugar de mantener el valor que tuviera seleccionado el usuario debido a cómo es creado (dependiente de la orgánica seleccionada).
Para solucionar este problema se hizo uso del método `onSaveInstanceState()` del cual ya se habló en mayor profundidad en la sección 2.3.8, que permite almacenar pares clave-valor y recuperarlos a través del objeto Bundle que se pasa en el argumento en el método `onCreate(Bundle)`. En este caso, lo que se salva en verdad es el estado de un fragment y no de una actividad, pero se ha hecho de forma similar a como se hace en una actividad. Se salva el índice como un valor entero (int) de la opción del spinner que tiene seleccionado el usuario, y en el método `onCreate(Bundle)` se comprueba si el objeto `Bundle` no es nulo primero. En caso de que no lo sea, se obtiene el valor del índice de la opción del spinner del periodo y se establece dicho índice como la opción que ha de estar seleccionada y se cargan los datos de dicho periodo en la pantalla.

- **Comunicación entre actividades**

La comunicación entre actividades se realiza a través de objetos `Intents`. Existen dos tipos de intents: implícitos y explícitos. Los intents implícitos son aquellos en los que se busca una funcionalidad y no una actividad de una aplicación en concreto. Por ejemplo cuando se desea abrir una página web en concreto se utiliza un intent para “abrir un navegador” pero no se especifica qué navegador en concreto. El sistema será el

encargado de utilizar el navegador por defecto o mostrar una lista al usuario para que elija el navegador que quiere utilizar, en caso de tener varios instalados en el dispositivo móvil. Los intents explícitos son aquellos en los que se llama a una actividad en concreto, generalmente de la propia aplicación para pasar a otra pantalla de la aplicación.
En esta aplicación se ha hecho uso de intents tanto implícitos como explícitos. En el caso de los implícitos se han utilizado por ejemplo a la hora de descargar la aplicación recomendada por el servicio de telefonía de la UC3M para el uso de la telefonía IP, donde se solicita descargar cierta aplicación del market por defecto del dispositivo móvil. También se han utilizado al solicitar el marcador telefónico para realizar las llamadas al CASO para poner una incidencia, o a la hora de abrir un navegador para descargar la normativa completa entre otros documentos. Los intents explícitos se han utilizado a la hora de pasar de una actividad a otra de la propia aplicación.

En algunos casos se necesita enviar información de una actividad a otra para realizar su función correctamente. Esta información se envía añadiendo unos extras al intent. En el caso del marcador del teléfono se añadía el teléfono que debía aparecer marcado en el marcador telefónico, o en el caso del navegador se le añade la URL con la dirección de la página web a abrir. Cuando se trata de información entre actividades de la misma aplicación, se pueden añadir además algunos tipos básicos de datos como strings (cadenas de caracteres), int (enteros), list (listas), etc. en formato clave-valor para recuperarlos en la actividad a la que se le envían. En el caso de necesitar enviar datos más complejos como pueden ser objetos (instancias) de alguna clase creada por el desarrollador en la aplicación, se pueden crear clases *serializables* o *parcelables* cuyos objetos se pueden enviar como un extra en el intent.
En la aplicación de telefonía corporativa de la UC3M es necesario el paso de datos complejos entre actividades cuando el usuario pulsa sobre una de las filas de las tablas de consumo para ver los consumos detallados. Para esta aplicación se ha hecho uso de la interfaz `Parcelable`, por lo que cada una de las clases que han de ser pasados sus objetos entre actividades implementan dicha interfaz. Si un objeto está compuesto de otros objetos complejos, las clases de dichos objetos han de implementar también la interfaz Parcelable hasta llegar a objetos que ya implementan la interfaz `Parcelable` o a tipos primitivos como `Strings`, `int`, etc.

3.7.4. Relativo a la interfaz del usuario

- **Uso de los componentes gráficos de Android y su modificación**

Durante el desarrollo de esta aplicación se ha hecho uso de múltiples componentes ya definidos en Android, algunos de ellos utilizándose tal cual como muchos campos de texto simples (`TextView`) pero modificando muchos de ellos para cumplir con las especificaciones gráficas que se requerían a esta aplicación. La mayoría de estos cambios sobre los componentes por defecto es el cambio de color del fondo de botones o

de campo de texto. En el caso de los botones o las opciones en una lista, como cuando se establece un color de fondo para una vista ese color se queda fijo, es decir se pierde la funcionalidad del cambio del color según el estado del botón (presionado, no presionado, con el foco, etc.). En estos casos, para mantener una experiencia de uso adecuada se crearon selectores de color en función del estado para los fondos para emular el comportamiento del cambio de color según el estado de la view (presionada, no presionada, etc.). El uso de este selector se puede ver en la Figura 47:

Figura 47 - Uso de selectores de color para el fondo de las opciones de las listas y botones: vista del botón en estado por defecto (izq.) y botón presionado por el usuario (dcha.)

En el caso de las listas se han creado layouts específicos para que algunas de ellas muestren colores de fondo e iconos para los menús en lugar de utilizar el layout de la lista por defecto. Se puede ver un ejemplo de las diferencias entre ambas listas en la Figura 48. También en esta figura se puede ver el degradado que se utiliza en el banner de la aplicación, creado en un fichero XML haciendo uso del elemento *shape* visto en la sección 2.3.3 de drawables.

Figura 48 - Diferencia entre lista personalizada (izq.) y lista simple por defecto (dcha.)

También se han modificado algunos campos de texto para dotarlos de apariencia de enlaces a páginas web tal y como suelen mostrarse en los navegadores web, como puede observarse en la Figura 49.

Figura 49 - TextView modificada para asemejarse a un enlace web

Otra modificación algo más compleja de un componente incluido en las librerías de Android ha sido la del spinner en la pantalla de consumo de grupo. Un objeto `Spinner` es un componente gráfico similar a un botón que cuando se pulsa sobre él aparece una ventana emergente (`Dialog`) con los posibles valores de ese spinner. Los spinner muestran por defecto la primera opción de la lista de opciones a elegir. Sin embargo, para esta aplicación se solicitó que se mostrara un texto cuando no se hubiera seleccionado ninguna opción de las orgánicas disponibles. Dado que los spinner no tienen por defecto esta opción a diferencia de los campos de texto editable `EditText` que sí la tienen, se ha poner como primera opción la palabra “Orgánica” ya que los spinner muestran por defecto la primera opción configurada en su `Adapter`. Un `Adapter` es un objeto puente, que provee de los datos y crea las views a mostrar en una view contenedora de datos variables. Sin embargo, se precisaba que esta opción “Orgánica” no apareciera luego en la ventana emergente con las posibles orgánicas disponibles. Para conseguirlo, a la hora de establecer el `ArrayAdapter` con los valores de las opciones y el layout a utilizar para la ventana emergente, se sobrescribe el método `getDropDownView()` para que en el caso de que si el spinner está en la posición 0 (es decir, en la posición por defecto “Orgánica”) al desplegar el menú con las opciones, esta primera opción se oculta estableciendo el tamaño de dicha View a 0 dp. En el resto de casos se muestra las opciones en la ventana emergente (ya que previamente se habrá ocultado la opción de “Orgánica” la primera vez que se pulsó sobre el spinner). En la Figura 50 se puede observar este comportamiento del spinner.

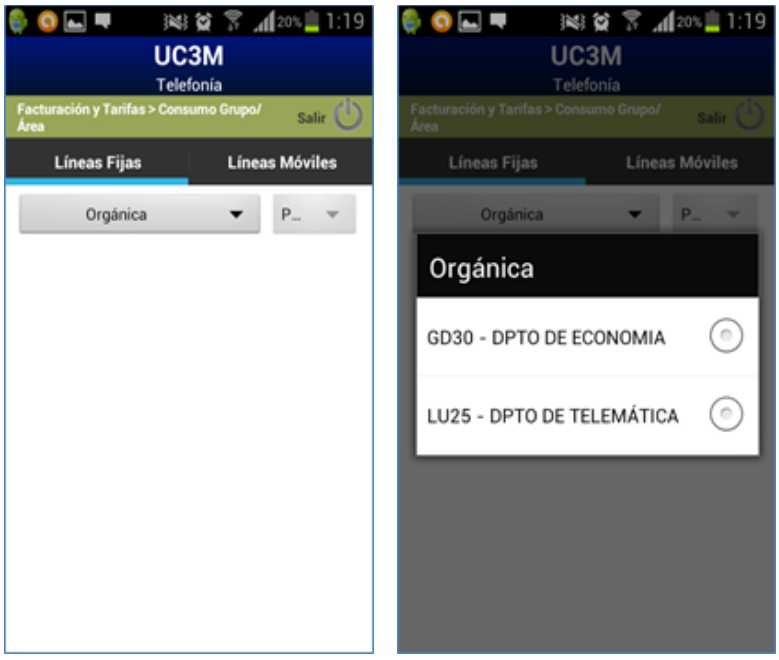

Figura 50 - Spinner mostrando el opción por defecto “Orgánica” (izq.) y la lista emergente con las opciones sin la opción por defecto “Orgánica” (dcha.)

- **Creación de nuevos componentes gráficos basados en otros ya existentes**

Para mostrar las imágenes de las guías de configuración y el manual de configuración de telefonía IP se creó un nuevo tipo de componente gráfico heredado de la clase `View`. Se trata de una imagen con funcionalidad "*pinch to zoom*". Esta funcionalidad es la típica utilizada en la galería de imágenes de Android que permite acercar la imagen (acercar el zoom) pulsando sobre la pantalla con dos dedos y separando los dedos, o bien alejar la imagen (alejar el zoom) pulsando sobre la pantalla con dos dedos y acercando los dedos. Además tiene otra funcionalidad añadida de acercar la imagen al máximo pulsando dos veces rápidamente la imagen y volviendo al tamaño original pulsando de nuevo dos veces rápidamente sobre la imagen (doble tap). Se decidió que la imagen sólo aumentaría hasta el doble de su tamaño original pues con este tamaño se ve claramente toda la información de ejemplo en las imágenes de las guías de configuración, y no se podría alejar el zoom más allá de su tamaño original. Para obtener un objeto de esta clase se crea el objeto o se referencia desde el XML, y se le establece un bitmap haciendo uso del método `setBitmap()`. Se puede convertir cualquier imagen de los recursos en un Bitmap haciendo uso del método `BitmapFactory.decodeResource()`.

- **Informar al usuario de forma visual**

Es importante informar al usuario visualmente de los procesos que se están llevando a cabo en la aplicación. En esta aplicación se han utilizado iconos de carga durante las operaciones que implican una cierta espera como conexiones a internet. Para ello se ha utilizado el icono de la Figura 51 como icono de carga:

Figura 51 - Icono de carga durante aplicaciones que implican cierto tiempo de espera

Otro ejemplo de información al usuario de forma visual en la aplicación es haciendo uso de Toast (mensaje que se superpone en la pantalla por un cierto periodo de tiempo, normalmente corto) para indicar ciertas funcionalidades como el acceso al detalle del consumo, tal y como se puede observar en la Figura 52.

Figura 52 – Uso de Toast

Cuando ocurran ciertos errores en la aplicación, se ha de informar correctamente al usuario, ofreciendo posibilidades de solucionarlo. Para esta funcionalidad se hace uso de ventanas emergentes llamadas *dialog* que se superponen a la pantalla que se estaba mostrando cuando sucedió el error. En el caso de esta aplicación se ha utilizado una subclase de `Dialog`: `AlertDialog` que simplifica alguno de los parámetros de configuración. Un `AlertDialog` permite establecer un título, un mensaje, y varios botones con opciones para el usuario: un positive button a la izquierda, un negative button a la derecha y un neutral button en el centro. Un ejemplo de esta funcionalidad se puede observar en la Figura 53. En la aplicación, el botón de respuesta positiva (positive button) siempre es "Reintentar" y el botón de respuesta negativa (negative button) "Cancelar". Pero el botón de respuesta (neutral button) dependiendo del tipo de error mostrará un botón de "Activar Internet" que llevará al usuario a la pantalla de ajuste de redes o bien un botón de "Llamar al CASO" que lleva al menú de Asistencia/Soporte de la aplicación en caso de que el error esté en los datos recibidos y el parseador de los archivos de respuesta de los web services de los que se obtiene los datos sea incapaz de obtener información correcta. Además, si el error se da en alguna de las pantallas del área de "Facturación y Tarifas" excepto la de inicio de sesión, al pulsar sobre "Cancelar" en el mensaje de error, aparecerá un botón en la pantalla con el texto "Reintentar" por si el usuario quiere volver a intentar obtener los datos posteriormente.

Figura 53 - Ejemplo de un AlertDialog

- **Creación de animaciones y patrón Data Drill Down**

A la hora de diseñar una experiencia de usuario más atractiva se decidió crear en algunas pantallas ciertas animaciones sencillas. Estas animaciones son de tipo `Tweened`, es decir, animaciones conseguidas a través de aplicar ciertos cambios sobre una imagen tales como rotación o traslación. Estas animaciones se han utilizado en el menú de la normativa de telefonía para conseguir el efecto del patrón Data Drill Down: la pantalla se desplaza hacia la izquierda o hacia la derecha para simular con la animación el entrar en un nivel dentro de la opción elegida en el menú anterior. Para realizar este efecto se ha hecho uso de la clase `ViewFlipper` que permite cargar distintas pantallas según la opción elegida por el usuario y de las animaciones creadas por código para las transiciones entre dichas pantallas. Por defecto al volver hacia atrás pulsando el botón de *atrás* (*Back)* del dispositivo móvil se volvería al menú principal de configuración y soporte, por lo que se sobrescribe el método `onBackPressed()` que maneja los eventos del botón *atrás* (*Back*) para que regrese a la pantalla anterior de la normativa en lugar de al menú de configuración y soporte, aplicando la animación de la transición correspondiente. Las diversas pantallas suelen ser listas de opciones que tienen su propio `Adapter` personalizado para añadir los iconos adecuados dependiendo de si es una lista con opciones seleccionables, o una lista final con información.
También se utiliza otro tipo de animación en la galería de imágenes deslizantes de las guías de configuración y del manual de configuración de VoIP en el que la imagen se vuelve por unos instantes transparentes en la transacción.

- **Uso de componentes gráficos más avanzados de las librerías de compatibilidad**

Android ha ido incorporando a sus librerías nuevos componentes gráficos útiles. Algunos de ellos han sido incluidos, aunque con ciertas limitaciones, en unas librerías de compatibilidad para que los desarrolladores puedan implementar estos nuevos componentes gráficos. En esta aplicación se han hecho uso de los siguientes entre otros: `Fragments`, `ViewPager` y `FragmentTabHost`.

De los fragments ya se habló en mayor profundidad en la sección 2.3.9 pero se puede pensar en ellos como una porción de funcionalidad incrustada en una actividad, generalmente asociada a una porción de la interfaz gráfica, que puede ser manejada con independencia del resto de porciones de interfaz. Estos fragments tienen así su propio ciclo de vida y manejo de eventos, pero siempre dependiente del ciclo de vida de la actividad que contiene dicho fragment. Un mismo fragment puede ser reutilizado en distintas partes del código. Se ha hecho uso de los fragments en la aplicación de este proyecto a la hora de crear pantallas dinámicas en función del tamaño del dispositivo móvil sobre el que se esté ejecutando. En las pantallas de consumo, se tendrán generalmente dos fragments, uno con la información de líneas fijas y otro de líneas móviles. En caso de que el dispositivo móvil en el que se esté ejecutando la aplicación sea una tablet u otro tipo de dispositivo con una pantalla de grandes dimensiones se dispondrán un fragment al lado del otro simplemente. En caso de ser una pantalla de tamaño reducido (handset) se utilizará el sistema de navegación por pestañas haciendo uso de la clase `FragmentTabHost` también incluida en las librerías de compatibilidad y a cada una de estas pestañas se le asociará uno de los fragments. Se puede ver esta disposición en la Figura 54, aunque en este caso las pestañas son “Consumo Voz”, “Tarifas Planas” y “Roaming”. El objeto de la clase `FragmentTabHost` debe utilizarse dentro de un tipo especial de actividad utilizada para los fragments: `FragmentActivity`.
En dicha Figura 54 se puede observar el uso de otro de los elementos utilizados de la librería de compatibilidad: el `ViewPager`. Un objeto de esta clase funciona como contenedor de páginas con datos (utilizados generalmente con fragments para controlar el ciclo de vida de cada página o con `Views` directamente). Se puede navegar entre las distintas páginas deslizando el dedo por la pantalla como si se quisiera pasar la hoja de un libro.
También en la Figura 54 se puede observar dicha funcionalidad en cada uno de los fragmentos para pasar entre las tarifas de “fijos” y las de “móviles”. También se hace uso de un `ViewPager` en las guías de configuración, para pasar de una imagen que muestra un paso a la siguiente o a la anterior.

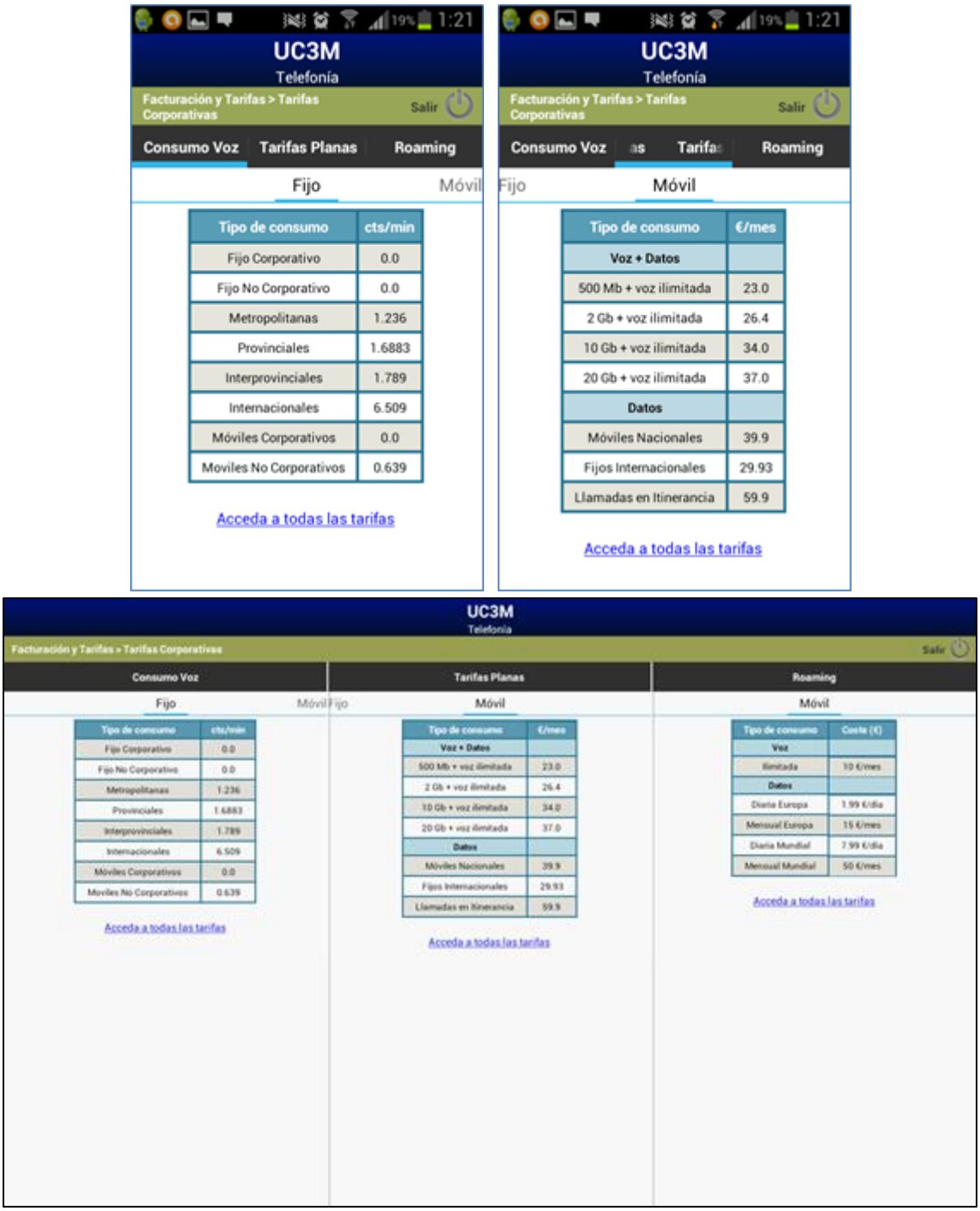

Figura 54 - Diferencia entre vista en un terminal handset (arriba) y una tablet (abajo) en Tarifas Corporativas

- **Peculiaridades de las tablas de facturación**

Las tablas de consumo creadas tienen cierta complejidad que no pudiera apreciarse a primera vista. En primer lugar, las tablas en Android (`TableLayout`) no tienen bordes internos ni externos, ni proporcionan métodos para implementarlos de forma nativa. Para

solucionar este primer requisito se pueden tomar dos opciones. La primera de ellas, más sencilla a priori, aunque menos flexible y menos elegante, sería dar un color de fondo a la tabla y que cada una de las `TextView` que muestran los datos tuviera otro color de fondo asociado y definiendo en sus atributos una separación para con el final del espacio que ocupa. Dicho espacio mostraría el color del contenedor de abajo, es decir el color que se le dio al fondo de la tabla creando el borde. Otro modo más elegante y fácil de modificar si fuera necesario es crear en un archivo XML para crear las celdas de la tabla utilizando para ello la etiqueta shape, que permite crear formas básicas como los rectángulos de las celdas y darles un color de relleno, de borde y un tamaño de borde entre otras características. Para el desarrollo de esta aplicación se utilizaron tres tipos de celdas generadas por XML fundamentalmente, un tipo para los títulos de la tabla, otro para las celdas de filas pares y otro último para las celdas de las filas impares.

Otra característica interesante de las tablas de consumo de los menús “Mi Consumo” y “Consumo de Grupo/Área” es su presentación cuando hay muchos datos y se ha de hacer scrolling (deslizar la pantalla para ver el resto del contenido). La tabla de consumo se ha diseñado de tal manera que los títulos de cada una de las columnas y el total del coste de ese periodo siempre sea visible como se puede observar en la Figura 55.

Figura 55 - La fila de coste total de un periodo con necesidad de scrolling (izq.) y sin necesidad de él (dcha.)

Tal y como está implementada la clase `ScrollView` en Android no permite tener más `Views` debajo ya que pasa a ocupar todo el espacio restante en la pantalla. Hay que valerse de algunos atributos de `ScrollView` y otros tipos de `Views` para conseguir dicho efecto. Hay que tener en cuenta el requisito real, es decir, que la fila con el coste total esté

en la última posición de la tabla, pero si la cantidad de datos es grande y hay que hacer scrolling esta fila debe estar siempre visible y hacer scrolling sólo de los datos de dentro. Por ejemplo, si el teléfono se pone en horizontal (landscape) sólo serán visibles 3 o 4 filas y se hará necesario hacer scrolling aunque haya pocos miembros en el departamento. Esto descarta una de las soluciones más comúnmente utilizadas en estos casos: el uso de un `RelativeLayout` (las vistas se pueden colocar en cualquier disposición a través de relaciones entre ellas como a la derecha de tal vista, o a la derecha del todo del contenedor padre) como contenedor raíz de todo el layout y establecer la fila del coste total pegado a la parte baja del contenedor padre. El problema radica en que la fila del coste estaría siempre en la parte de abajo en cualquier caso, tanto si hubiera que hacer scrolling por ser muchas filas con datos como si no fueran muchas filas en cuyo caso quedaría un hueco grande entre el filas de los datos de la tabla y la fila del coste. Si se intenta colocar la fila del coste total justo debajo del `ScrollView` no funciona como sería pensable, ya que limita el espacio del `ScrollView` a dos o tres filas, pone la fila del coste total y deja un hueco enorme debajo. Finalmente la solución implementada (aunque posiblemente no la única que haya, ni la más sencilla) es crear un `TableLayout` (clase contenedora para crear tablas en Android) que contenga un `ScrollView`. Este a su vez contiene otro `TableLayout` con los datos de los miembros del departamento en el caso de la pantalla de “Consumo de Grupo/Área” o las líneas de teléfono del usuario en la pantalla de “Mi Consumo”, sobre los cuales serán los que se pueda hacer el scrolling para ver más datos, y ya fuera del `ScrollView` pero dentro aún del primer `TableLayout` se añade la fila con el coste total como un `LinearLayout` dentro de un `TableRow` (fila de un `TableLayout`). Además, las alturas del `ScrollView` y del `LinearLayout` que contiene la fila del coste total han de tener un tamaño 0 y se le establece el atributo weight a 1 a ambos, para que el valor del tamaño de la altura de cada uno de los componentes se calcule en función de los pesos que se les establezca.

Ya que el ancho de las columnas es variable según el tamaño de información que contiene y del ancho de la pantalla del dispositivo en el que se ejecuta, y dado que se quiere mantener tanto los títulos de las columnas como la fila con el coste total del periodo siempre visibles en la pantalla, en el código no forman parte de la tabla con los datos de consumo de los miembros del departamento o área. Al no formar parte de la tabla de datos que es la que rige el tamaño que tienen las columnas y al ser este ancho dinámico en función de los datos y pantalla del dispositivo, los anchos de las columnas de los datos no coinciden con los de los títulos ni con la del coste total del periodo. Para solucionar este inconveniente se hace uso del listener (escuchador) `ViewTreeObserver.OnGlobalLayoutListener()` el cual se le aplica a la última fila de la tabla de datos. En realidad da igual sobre qué fila de la tabla de datos escoger ya que éstas al estar dentro de la misma tabla sí tendrán todas el mismo tamaño. Este listener lanza un método *callback* cuando el estado de la interfaz gráfica o la visibilidad de las vistas cambia, por lo que cuando el interfaz cambie y el tamaño de la anchura de los elementos de la última fila de la tabla de datos sea mayor que 0, significará que ya se le ha asociado el tamaño para dicho dispositivo. Entonces se toman los tamaños de cada una de las columnas de la última fila de datos y se aplica al ancho de cada una de las

columnas de los títulos. En la fila del coste total del periodo sólo hay dos columnas por lo que la anchura de la primera columna la conformará el tamaño de las dos primeras columnas de la tabla de datos. La celda con el valor del coste total del periodo tendrá el mismo ancho que el resto de celdas del coste de cada una de las líneas de datos.

Por último, para dar una mejor experiencia de uso con respecto a las tablas de consumo se ha hecho que cuando el usuario pulse sobre una de las celdas de la tabla, toda esa fila cambie de color mientras se esté pulsando. Este comportamiento que a priori pudiera parecer sencillo utilizando un selector de color como en otros botones, entraña algo más de complejidad. Hay que tener en cuenta que la fila está compuesta por las celdas en su totalidad así que cuando una es pulsada, si se utiliza un selector de color en función del estado como en otras ocasiones sólo se cambiaría el color de dicha celda, pero no el de las otras dos. Para obtener el comportamiento que se requiere se hace uso del listener `OnTouchListener()` que se le aplica a cada una de las celdas. Cuando el usuario interacciona con alguna de las celdas se produce una llamada al evento *callback* `OnTouch(View, MotionEvent)`. Las llamadas a este método no se producen sólo al pulsar, sino en cada uno de sus estados: pulsar la pantalla, levantar el dedo de la pantalla, o cada vez que se mueve el dedo por la pantalla si se había pulsado previamente en la celda. Estos estados vienen definidos en el objeto `MotionEvent` del argumento del método *callback* `OnTouch(View, MotionEvent)`. Al fondo de cada una de las celdas se le aplica un selector de color según el estado. En el método `OnTouch()` si se trata del `MotionEvent` de pulsar la `View` (`MotionEvent.ACTION_DOWN`) se hace que el estado de todas las celdas de la fila a la que pertenece la celda pulsada cambien su estado a presionado haciendo uso del método `setPressed(true)`. Cuando se cancela la pulsación, se mueve por la pantalla el dedo sin haber soltado antes la pulsación (levantado el dedo de la pantalla), o se levanta el dedo de la pantalla todos las celdas ponen su estado de presionado a false y pasa a ejecutarse el *callback* `OnClick(View` en el cual se crea el intent para la pantalla de detalles.

- **Listas expandibles multinivel**

Las listas expandibles multinivel no se han utilizado finalmente en este proyecto, sin embargo se ha considerado incluirlo en la presente memoria por el interés de su desarrollo aunque finalmente se descartara este modelo por uno que mostrase menos información al usuario de golpe, ya que creaba un diseño confuso. Se probaron para la funcionalidad del “Asesor de Tarifa” para navegar entre los distintos niveles de información. Una lista expandible es aquella en la que al pulsar sobre una de las opciones de la lista, aparece una nueva lista debajo de la opción seleccionada pero dejando ver el resto de opciones que no fueron utilizadas más abajo de la nueva lista de opciones. Al pulsar sobre la opción seleccionada de nuevo, el listado de segundo nivel se encoge y desaparece. El problema que surge a la hora de crear una lista expandible es que la incluida en las librerías de Android sólo soporta hasta 2 niveles de profundidad y las listas que se precisaban para el Asesor de Tarifas han de ser de 5 o 6 niveles. Para ello se

desarrolló una lista multinivel de forma que al seleccionar en una opción de la lista no aparecía otro listado común, si no otra lista expandible, que a su vez, si se elegía una nueva opción contenía otra lista expandible, etc. Las listas expandibles simples de Android ya son complejas de por sí pues se han de crear a través de objetos `Map<K,V>` o bien creando clases que hereden de `Adapter`. Se decidió crear adapters propios ya que también se necesitaban que cambiaran los colores de fondo, el tamaño de las filas y algún detalle más. Sin embargo al dejar todas las opciones abiertas, aunque se podían comparar de forma rápida unas con otras, se mostraba muchísima información al usuario en pantalla, por lo que transmitía más confusión que ayudarle. Al final se decidió utilizar el desarrollo que tiene actualmente el "Asesor de Tarifa" el cual se explica a continuación.

- **Asesor de Tarifa**

El "Asesor de Tarifa" aunque a primera vista parecía sencilla, finalmente ha resultado un desarrollo bastante complejo. La funcionalidad se basa en guiar al usuario hasta la tarifa que más le conviene a través de opciones que se le van mostrando. En un primer momento se intentó utilizar la variante de lista expandible multinivel explicada en el punto anterior, sin embargo al seguir mostrando todas las opciones que el usuario hubiera seleccionado previamente, el resultado fue una interfaz confusa y poco manejable. Se quería que las opciones que no se habían seleccionado se escondiesen. Además se quería que se pudiera volver a las opciones anteriores sin tener que pasar por todas las opciones de nuevo desde el principio. Por lo tanto, se decidió que presionando en alguna de las opciones previas que se eligieron, todas las seleccionadas después desaparecen y vuelven a mostrarse las otras opciones que estaban al mismo nivel de opción, similar al funcionamiento de las listas expandibles comunes.
Este funcionamiento implica tener una serie de `TextViews` con sus propios ID para averiguar sobre cual se había pulsado. La solución más sencilla hubiera sido crear toda la jerarquía en un layout y asignarles a cada View su ID, su visibilidad y su comportamiento y colores a través de los atributos de `TextView` en el propio layout. Cada nivel de opción tiene diferente número de opciones, aunque en la mayoría de los casos sólo se tratan dos opciones por nivel. Además cada opción de un mismo nivel puede generar un nuevo nivel de opciones o el nombre de la tarifa recomendada. Sin embargo crear las views requiere un gran cómputo por lo que tardaría bastante en cargar una jerarquía tan extensa y compleja creando una mala experiencia de uso, teniendo en cuenta además que la mayoría de las views nunca se van a ver porque no van a elegir esa rama de opciones, además de mucha repetición de código innecesaria.

Se decidió crear un layout básico en el que se incluía el primer nivel de opciones. Por otro lado se creó otro layout con dos opciones que era el caso más utilizado dándole ya un ID a cada una de las opciones lo que simplificaba el código. De este modo, no se requiere un ID concreto para cada una de las opciones de cada uno de los niveles de opción que tienen dos opciones, sino que sólo se comprueba el nivel en el que está y si es la primera o la segunda opción. Además para los niveles de opción en los que hay varias opciones,

como no siempre son el mismo número de opciones, se creó otro layout con una única opción. Por código se crean tantas instancias de dicho layout como sea necesario y se le asigna un ID por código. Estos IDs se crean en un archivo de recursos XML de IDs a partir del cual el sistema Android crea un número identificativo único. De este modo, el programador no se tiene que preocupar de definir un número que sea único en la aplicación. Por último se crea otro layout para las `TextView` finales pues muestra otro color, no tiene icono asociado, etc. La mayor parte de los atributos de estas `TextView` están definidos en el archivo XML de style (estilos) y se les asocia dicho estilo directamente para limitar el uso de líneas repetidas en los layouts y crear un layout más limpio y fácil de modificar en todas las `TextView` a la vez. Además todos los datos con los textos de las opciones están definidos en un archivo XML de arrays y strings para que sean fácilmente modificables.

Para hablar sobre la creación de la jerarquía por código se utilizará directamente un ejemplo para mayor comprensión del lector. Cuando el usuario pulsa sobre la primera opción del primer nivel de opción, el cual tiene dos opciones, cambia el icono de la opción que se pulsó a una flecha hacia arriba, se muestra un segundo nivel de opciones y desaparece la segunda opción del primer nivel de opciones, pero no se elimina de la jerarquía, sólo se hace invisible para que se pueda volver a ella de formar rápida y sin tener que volver a recrearla. Este segundo nivel tiene dos opciones y el fondo es color blanco con el texto en negro y con el icono de una flecha hacia abajo tal y como suele mostrarse en las listas expandibles. Cuando el usuario ha pulsado sobre la primera opción del primer nivel, como el siguiente nivel tiene dos opciones, se crea una instancia (se infla) el layout de dos opciones y se añade al `LinearLayout` que contenía a las dos primeras opciones, lo que se puede llamar como el contenedor del primer nivel de opciones. El segundo nivel también está contenido dentro de un `LinearLayout` que sirve como contenedor del segundo nivel de opciones. Cada opción de cada nivel tiene asociado un escuchador en función del nivel de opciones al que pertenezca.
Cuando se pulsa sobre una de las opciones se invoca al método `onXXLevelClicked(View)` siendo XX el nivel de opciones al que pertenezca. Este método se encarga de gestionar en qué posición del `LinearLayout` de ese nivel se ha de añadir el siguiente nivel en función de si era este nivel un nivel de dos opciones o de opción múltiple. En el caso del nivel de dos opciones es sencillo averiguar en qué posición se ha de colocar el siguiente nivel filtrando por el id de las opciones. En el caso de un nivel de opción múltiple, a través del argumento View que se le pasa al método con la View que fue seleccionada, se puede obtener la posición de dicha `View` en el `LinearLayout` contenedor de ese nivel, y colocar el siguiente nivel en una posición más a la que ocupa la View de la opción que se seleccionó. Con el manejo de la visibilidad de las `Views` pasa algo similar, en el caso de que el nivel fuera de dos opciones se filtra simplemente por el id de las dos opciones y el que no ha sido seleccionado se pone como invisible. Si era un multinivel se obtiene la posición utilizando el argumento View que se le pasa a la función a través del método `ViewGroup.indexOfChild(View)` y se hacen invisibles todas las vistas previas a ese índice y todas las posteriores a esa posición + 1, ya que se ha colocado ya el siguiente nivel en la posición siguiente a la de la View

seleccionada. Se les definen un ID a los `LinearLayout` que contienen a los nuevos niveles además de añadirle los IDs que les correspondan si el siguiente nivel es de más de dos opciones ya que como ya se vio no tenían ninguno definido por defecto. Estos IDs son necesarios si el usuario decide volver a una de los niveles de opción previos saber a cuál tienen que volver, y en el caso de niveles de más de dos opciones para saber qué opción fue elegida.

3.7.5. Obtención y tratamiento de datos

- **Petición POST, esquema HTTPS y expire time**

En las pantallas de "Facturación y Tarifas" se utilizan datos sensibles para el usuario, sobre todo en el inicio de sesión del usuario, en el que además del nombre del usuario se envía la contraseña que utiliza en su cuenta de la universidad. Por ello se ha de enviar dicha información de forma cifrada y por una conexión segura. Una petición consiste en enviar la información sensible en el cuerpo de la petición HTTP de forma cifrada. Para ello, se crea una Lista de objetos `NameValuePair` siendo las claves las variables que espera el web service. Se crea un objeto `Entity` pasando por parámetro un objeto `UrlEncodedFormEntity` que se crea pasando por parámetro a su vez la lista de objetos `NameValuePair` con el usuario y la contraseña. Este objeto se añade al objeto `HTTPClient`. Este cliente es una instancia de una clase creada en la aplicación para asegurar que se utiliza el protocolo seguro HTTPS.
Utiliza un `DefaultHttpClient` con una configuración específica. Esta configuración se basa en un esquema de registro utilizando el protocolo HTTPS y la utilización de una coneción SSL (Secure Socket Layer) mediante la factoría de `SSLSocket` de Android. Además se le añade un *expire time*, o tiempo de expiración de la conexión, en los parámetros de configuración para que en caso de que una conexión tarde más de 8 segundos en crearse, se informe al usuario de un error en la conexión. Este tiempo de expiración fue necesario ya que se podía dar el caso de que el usuario se conectase a una red WI-Fi en la que haya que aceptar las condiciones de uso en el navegador, como Wi-Fi UC3M, y que no se hubieran aceptado aún. Aunque pasa la comprobación de que el usuario esté conectado a internet, no tiene conexión real, y por lo tanto la petición no llega a establecer conexión con el web service quedándose en estado de validando al usuario indefinidamente.

Al ser esta seguridad de las contraseñas un mecanismo muy importante en la aplicación se comprobó que efectivamente la información se enviaba a través de una conexión HTTPS. Además se comprobó también que en la URL no se enviaba los datos de inicio de sesión si no en el cuerpo de forma codificada utilizando el emulador del SDK de Android y el programa Wireshark (antiguamente Ethereal), un analizador de protocolos de red que permite ver el tráfico en una red filtrando la información por numerosas opciones, como por ejemplo la IP del origen, del destino, MAC, etc.

- **Parser**

Un parser es una analizador sintáctico que permite obtener los datos de un documento de texto codificado en algún formato para su uso en la aplicación. En la aplicación de telefonía corporativa de la UC3M se han necesitado varios parsers: uno para procesar los datos en formato XML que devuelven los distintos web service del área de "Facturación y Tarifas"; otro para el texto de los avisos del menú principal; y otro para la respuesta del web service del inicio de sesión. Existen tres posibles parsers de XML en Android: XmlPullParser, DOM y SAX. Para esta aplicación se decidió utilizar el `XmlPullParser` ya que es el que recomiendan utilizar en la página web oficial de desarrolladores Android (Android Developer [22]), además de ser más simple y más intuitivo que DOM. Además, los archivos XML a parsear no tienen una estructura compleja.
`XmlPullParser` recorre el documento de texto de evento en evento hasta finalizar el documento, obteniendo la información de los campos que interesan al desarrollador e ignorando el resto de los datos. Se obtiene el tipo de evento (`START_TAG, END_TAG, TEXT,...`) y si se trata del tipo de evento que se quería se obtiene el nombre del evento y se comprueba si es alguno de los que interesan para la aplicación, en cuyo caso se obtiene el valor de alguno de sus atributos o del texto a continuación de la etiqueta. Al cerrar la etiqueta de algunos campos importantes que se reutilizan al haber varios como el de </periodo> se procesan todos los datos para añadirlos al objeto que finalmente se devuelve a la actividad que utiliza el parser. Ésta utilizará la información contenida en dicho objeto para mostrar los datos en la pantalla al usuario.

- **SharedPreferences**

La clase `SharedPreferences` permite almacenar de forma persistente valores de tipo clave-valor primitivos. Estos datos se almacenan en la memoria privada de la aplicación. Puede seleccionarse si se desea que otras aplicaciones puedan acceder a estos datos pero sólo en modo lectura, en modo lectura y escritura o que sean privados para la aplicación. En esta aplicación se ha utilizado la clase `SharedPreferences` para almacenar el nombre de un usuario que ha iniciado sesión en la zona de "Facturación y Tarifas" y que ha elegido almacenar dicho nombre para utilizarse en el autocompletado del usuario en futuros inicios de sesión. Siguiendo uno de los requisitos de la aplicación, la contraseña no es almacenada en el autocompletado, y el usuario tendrá que introducirla cada vez que quiera iniciar sesión. No es necesario encriptar la información del nombre de usuario ya que éste es público en múltiples apartados de la web de la UC3M. Además, ya cuenta con la seguridad básica de almacenar la información en modo privado.
Cuando el usuario introduzca las credenciales de su usuario y contraseña y éstas se validen, se busca si el nombre que se introdujo ya estaba en la lista de nombres almacenados. En caso de estar ya ese nombre almacenado pasa directamente a la siguiente pantalla. En caso de tratarse de un nuevo nombre de usuario se muestra un cuadro de diálogo preguntándole al usuario si quiere almacenar este nuevo nombre de

usuario para el autocompletado, si borrar todos los nombres existentes o bien si no quiere guardarlo y continuar hacia la siguiente pantalla.
Como la clase `SharedPreferences` sólo almacena tipos primitivos de datos y cadenas de caracteres (strings), se decidió guardar todos los nombres como una único string separando cada nombre con el carácter de escape ";".

3.7.6. Otros aspectos relativos al código

- **Thread o AsyncTask**

Como ya se habló en la sección 2.3.12, al ejecutarse una aplicación el sistema crea un thread (hilo) de ejecución especial denominado UI Thread o *main*, en el que se ha de realizar todas las operaciones que tengan que ver con la interfaz gráfica. Como este hilo se encarga de procesar todos los eventos de la interfaz gráfica no se puede bloquear más de 5 segundos o el sistema dará por hecho que la aplicación se ha quedado bloqueada creando una mala experiencia de uso. Se mostrará entonces un cuadro de diálogo al usuario permitiéndole que pueda cerrar la aplicación que no está respondiendo adecuadamente. Por esto, toda aplicación que precise de operaciones de larga duración como la obtención de datos de Internet se ha de realizar en un nuevo hilo. También se puede hacer uso de la clase `AsyncTask`, la cual provee de un hilo en segundo plano y de mecanismos para informar al usuario que se ejecutan en el `UI Thread`. Se han utilizado hilos para operaciones en segundo plano o clases que heredan de la clase `AsyncTask` en multitud de ocasiones durante el desarrollo de la aplicación. Por ejemplo, en todas las pantallas en las que se implique un acceso a Internet para obtener información como en el área de "Facturación y Tarifas" o en la obtención del mensaje de los avisos de telefonía o a la hora de parsear las respuestas de los web services pues pueden contener gran volumen de datos, sobre todo la respuesta del web service de "Consumo de Grupo/Área". Para informar al usuario se utilizan los gráficos de barras circulares de carga por defecto de las librerías de Android.

- **Comprobación de conexión a Internet**

Antes de realizar cualquier petición o consulta a través de Internet se comprueba si el dispositivo móvil tiene conectividad a Internet haciendo uso del servicio `ConnectivityManager`. Para poder acceder a este servicio se necesita haber especificado en el archivo AndroidManifest.xml que se necesita el permiso `ACCESS_NETWORK_STATE`. Si el dispositivo móvil tiene acceso a Internet se continúa con la petición, si no se muestra un mensaje de error al usuario que le permite volver a intentar obtener los datos de Internet (cuando se trata de pérdidas de conectividad temporales por túneles, etc), ir a los ajustes de redes para activar el Wi-Fi o los datos de

paquetes, o bien cancelar y no realizar ninguna acción. En caso de seleccionar “Cancelar” se mostrará un botón de Reintentar en la pantalla por si quiere volver a intentar acceder.

- **Uso de adapters personalizados**

Para el desarrollo de esta aplicación se han utilizado dos tipos de adapters propios, uno genérico para las listas presentes en la aplicación y otro para las listas de la pantalla de “Normativa de Telefonía”.
Para las listas más comunes en la aplicación se creó un `Adapter` que fuera lo más genérico posible para que fuese válido tanto en los casos en los que la lista ha de tener un color de fondo alternando según la posición que ocupe en la lista como en los que además se ha de añadir un icono, valores que pasan como argumento al constructor. Hereda de la clase `ArrayAdapter<String>`, y toda la lógica va en el método `public View getView(int position, View convertView, ViewGroup parent)` el cual ha de ser sobrescrito. Este método es llamado para cada una de las posiciones del adapter para la lista y ha de devolver la View que se mostrará al usuario en dicha posición.
Para las listas de la pantalla de “Normativa de Telefonía” se creó un nuevo tipo adapter para que simplemente se pasara en el constructor el texto a escribir en cada una de las opciones. También se ha de pasar al constructor si son elementos “clickables”, es decir, si son listas que han de mostrar un nuevo nivel del `ViewFlipper` (para más información consultar el apartado “Creación de animaciones y patrón Data Drill Down” de la sección 3.7.4) en cuyo caso son objetos que pueden recibir un “click” y por lo tanto el argumento clickable ha de ser “true”. Si son elementos de una lista finales sin que inicien nuevos niveles de profundidad el argumento clickable ha de ser “false”. También tiene un icono distinto asociado en función de si el objeto es clickable o no.

- **ExpendingBaseActivity**

Esta clase se creó durante un proceso de refactorización de la aplicación. Es la clase base de la que heredan casi todas las clases de “Facturación y Tarifas” y que reúne el comportamiento similar de todas ellas. En todas las clases que componen el área de Facturación y Tarifas excepto en la pantalla de inicio de sesión y en el asesor de tarifas han de tener los métodos `loadData()`, `parseData()` y el objeto `AlertDialog` encargado de mostrar los errores al usuario que se puedan dar durante los métodos `loadData()` y `parseData()`. Se trata de una clase abstracta, cuyos métodos `loadData()` y `parseData()` son abstractos y están sin implementar ya que difiere en cada una de las clases que heredan de esta clase la manera de implementarse estos métodos. El método que crea el `AlertDialog` en función del tipo de error sí está implementado pues se trata del mismo código en todos los casos de las subclases.

- **Clase Constants**

Se trata de una clase compuesta por variables primitivas, `Strings` y `enums` (enumeraciones), todas ellas estáticas y finales (excepto las enums que ya son finales de forma implícita). Contienen todas aquellas constantes utilizadas en la aplicación que no son dependientes del idioma o del tamaño de pantalla, etc para que no tengan que ser incluidas en los archivos XML de recursos. Al estar todas las constantes en una misma clase permite que puedan ser encontradas para ser modificadas fácilmente y además permite que sean reutilizadas en distintas clases. Al ser estáticas (static) son variables de clase que pueden ser referenciadas sin crear una instancia de la clase que las contiene si no llamándolas sobre la propia clase, por ejemplo `Constants.LANDLINE_TYPE`. Algunos ejemplos de las constantes incluidas en esta clase serían todas las etiquetas de los archivos XML de respuesta de los web service, necesarias durante el parseo de los datos o las URL de los web service.

- **Patrón de diseño Singleton**

Se decidió que para la creación de la sesión del usuario en la aplicación se utilizaría el patrón de diseño Singleton, ya que es necesario que sólo exista una única instancia de dicha clase en la aplicación. El patrón Singleton se basa en que la clase tiene una variable del tipo de esa misma clase (en el caso de esta aplicación se la ha denominado `UserSession`) y el constructor de la clase es privado. La instancia de la variable de tipo `UserSession` se obtiene desde un método de clase (utilizando la palabra reservada `static`) que comprueba previamente si ya existe una instancia de la clase. En caso de que la instancia no sea null, es decir, ya existiera, se devuelve la instancia ya existente. En caso contrario se crea una nueva instancia y se devuelve dicha instancia. Cuando el usuario sale de su sesión la instancia de `UserSession` es puesta a `null` nuevamente, y por lo tanto destruida.

- **Nombres de tarifas dinámicos**

Los nombres de las tarifas y sus costes suelen variar varias veces por año al menos. Se ha intentado que el usuario tenga que instalar las menores actualizaciones posibles y que la aplicación tenga por lo tanto esta información actualizada sin depender de que el usuario haya actualizado la aplicación. Para ello, se decidió que la información de las tarifas fuera dinámica, es decir, serían otro campo más a incluir en los XML de facturación y tarifas. El principal problema es que implica tener una jerarquía de clases compleja en la que almacenar la información de modo que no siga una estructura fija, sino adaptable a la información disponible en los archivos XML de datos. Otra ventaja, es que ahorra tiempo y esfuerzo al evitar tener que cambiar la estructura de los parseadores y las etiquetas asociadas cada vez que se cambien éstas.

- **Esconder el teclado antes de llamar al método finish()**

La necesidad de tener que esconder el teclado antes de llamar al método `finish()` radica en un problema con la actividad de autenticación de los usuarios. Sin embargo, fue un bug complejo de descubrir ya que la consecuencia visible estaba en la siguiente pantalla, cuando se cargaba la lista del menú de "Facturación y Tarifas".
El bug consistía en que a veces al entrar en ese menú, la lista estaba descolocada y el nombre del primer menú aparecía en último lugar, incluso a veces el segundo nombre también. Sin embargo a veces estaban bien colocados, o al entrar en alguno de los menús y volver a salir se colocaban bien. Siempre se pensó que tenía que ver con que la actividad anterior, la de autenticación, pues inicia una nueva pila de tareas, llama al método para que comience la siguiente actividad (la del menú de "Facturación y Tarifas") y posteriormente llama al método `Activity.finish()` para que sea eliminada de la pila de actividades y al dar al botón de atrás ("*Back*") desde el menú de "Facturación y Tarifas" no pase de nuevo por la pantalla de autenticación si no que regrese de nuevo a la pantalla de inicio de la aplicación. Al llamar al método `finish()` se pensaba que provocaba que se volviera a cargar parcialmente la siguiente actividad mostrando alterados los valores de la lista, ya que si no se llamaba a este método y simplemente se llamaba al de iniciar la siguiente actividad, la lista se cargaba siempre bien.

Tras muchos cambios en el código que no consiguieron solventar este bug, se descubrió que lo que provocaba esa alteración era que el teclado tardaba unos instantes más en cerrarse que la actividad de autenticación. Por tanto, era el teclado el que provocaba la alteración en el orden de la lista de "Facturación y Tarifas". Finalmente el bug se corrigió de tal forma que cuando el usuario pulsa el botón de "Aceptar" para intentar la autenticación en la aplicación, se induce por código el que el teclado se cierre en ese momento haciendo uso del siguiente código:

```
// Hide the soft keyboard
InputMethodManager imm = (InputMethodManager)
LoginActivity.this.getSystemService(Context.INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(mPasswordView.getWindowToken(),
InputMethodManager.RESULT_UNCHANGED_SHOWN);
```

3.8. Refactorización

Se llevaron a cabo algunas fases de refactorización del código durante el desarrollo de esta aplicación. Una refactorización implica realizar cambios en el código o en su estructura para optimizar, simplificar el código, eliminar duplicidades y obtener así un código más limpio y fácilmente modificable, pero siempre sin añadir nueva funcionalidades a la aplicación. El proceso de refactorización se lleva a cabo tras implementar cada nueva funcionalidad en la aplicación o cuando se ve necesario realizar

alguna que afecta a mayor número de funcionalidades tales como la reestructuración de paquetes. A continuación se hablará de algunos cambios significativos en algunas refactorizaciones:

- Crear nuevos paquetes y renombrarlos para un mayor entendimiento de la estructura de paquetes del código.

- Renombrar algunas clases y moverlas a otros paquetes más adecuados.

- Crear archivos XML específicos para los textos del asesor de tarifas y de la normativa de telefonía en forma de string-array y de string para que estén todos los textos de cada tipo juntos y sea más fácil de localizar si es necesario algún cambio en alguno de los textos. Cambiar los string-array de un sólo elemento por un string simplemente.

- Los `ProgressDialog` que se utilizaban para mostrar un mensaje de "Obteniendo los datos" en un cuadro de texto se sustituyeron por un icono simple que gira para mostrar el estado de "cargando" en la aplicación. Sigue teniendo la misma funcionalidad final de informar al usuario, pero de este modo queda una interfaz más limpia, clara y actual.

- Quitar threads que eran innecesarios en las actividades del área de "Facturación y Tarifas".

- Crear la clase abstracta `ExpendingBaseActivity` de la que heredan casi todas las clases del área de "Facturación y Tarifas" como ya se habló de ello en el apartado `ExpendingBaseActivity` de la sección 3.7.6.

3.9. Problemas y tareas a afrontar por el desarrollo multidispositivo

Como ya se ha hablado en varios ocasiones a lo largo de la presente memoria, se ha de tener un especial cuidado a la hora de desarrollar una aplicación en Android a la gran variedad de dispositivos móviles que existen con Android, cada uno con sus dimensiones, calidades de pantalla, distintas resoluciones, etc. Además no todos los dispositivos Android tienen la misma versión del sistema operativo, lo que se traduce en distintos iconos básicos de Android, y otros tipos de problemas con los que se ha tenido que lidiar a la hora de realizar la aplicación de telefonía corporativa de la UC3M. Este problema es conocido por fragmentación en Android.

A continuación se va a hablar sobre algunos de los problemas o dificultades que se han tenido a la hora de desarrollar la aplicación de telefonía corporativa de la UC3M, catalogándolos por problemas por las características físicas de los dispositivos y problemas por las distintas versiones de Android.

En la Tabla 3 se hace un pequeño resumen de estos problemas y dificultades.

Tabla 3 - Resumen problemas y dificultades por la fragmentación

Problema o dificultad	Descripción	Debido a...
Diseño para handset y tablet en una misma aplicación	Utilizar fragments para un diseño dinámico que sirva tanto para handset como tablets	Características físicas del tipo de dispositivo móvil
Guías de configuración adecuadas	Utilizar las imágenes de la guía de configuración del Galaxy SII si se ejecuta en handsets y las de la Galaxy Tab si se ejecuta en tablet	Características físicas del tipo de dispositivo móvil
Colores diferentes en diferentes pantallas	Cada dispositivo móvil tiene su gama de colores de pantalla, por lo que un mismo color hexadecimal se ve distinto en una pantalla u otra	Características físicas del tipo de dispositivo móvil
Necesidad de diferente tamaño las letras del banner	En handset el tamaño de la letra del banner ha de ser más pequeño de lo normal, pero en tablet hay que agrandarlo para que no se vea demasiado pequeño	Características físicas del tipo de dispositivo móvil
No repetir los mismos archivos XML alternativos para todos los distintos tipos de pantalla	Hay tipos de pantalla que usarán el mismo layout pero que van en carpetas de recursos alternativas distintas según la versión de Android	La versión de Android
Tamaño de letra en las listas distinta	Las listas tienen por defecto un tamaño distinto según las versiones de Android	La versión de Android
Gráficos distintos	Algunos elementos gráficos de las librerías de Android cambian de forma y color según la versión de Android	La versión de Android
El mensaje de setError() no se ve	Debido al cambio de colores primarios el mensaje de error de los EditText no se ve en versiones de la 3.0 en adelante	La versión de Android
Las redes se activan en distintos sitios	Las red wifi y la red de paquetes ya no se activan en la misma pantalla a partir de la versión 3.0	La versión de Android
Querer usar elementos gráficos introducidos en versiones posteriores	Querer usar elementos de navegación o gráficos útiles y modernos en versiones antiguas	La versión de Android
Problemas con el divisor de las pestañas	El método setDividerDrawable() de TabWidget no funciona de igual manera para todas las versiones de Android	La versión de Android
Crear las pestañas de navegación para toda las versiones	No existe la ActionBar en las versiones anteriores ni en las librerías de soporte para poder implementar de forma fácil las pestañas de navegación	La versión de Android

Esta aplicación fue pensada para que se pudiese ejecutar en la mayor cantidad posible de dispositivos móviles, pero a su vez, centrándose sobre todo en que funcionara sin problemas y adaptándose algo más a los dispositivos homologados por la universidad. A continuación se expone una lista de los dispositivos homologados por la universidad, describiendo si es un handset o un tablet y la versión del sistema operativo que tenían cuando se implementó la aplicación. Algunos de estos dispositivos se han actualizado a versiones posteriores, pero ya no se dispone de ellos para probar la aplicación:

- Galaxy S2: handset, versión 4.1.2
- HTC Desire HD: handset, versión 2.3.5
- Galaxy Tab 7.0”: tablet, versión 4.0
- Galaxy Tab 10.1”: tablet, versión 4.0.4

Algunas tareas a afrontar y problemas relativos al dispositivo físico durante la realización de este proyecto han sido los siguientes:

- Diseñar la interfaz para tablets y handsets utilizando fragments. Se controla por código el tipo de dispositivo móvil según el layout que esté cargando el sistema en la ejecución de la aplicación. Si se está utilizando el layout para handset y se han de colocar los fragments en pestañas haciendo uso del `FragmentTabHost`. Si se está utilizando el layout para Tablet se han de colocar los fragments uno al lado del otro para mostrar ambos a la vez. Cualquiera de las pantallas de consumo excepto el asesor de tarifas sirve como ejemplo. Algunas de ellas se muestran en las Figura 56 y Figura 57.

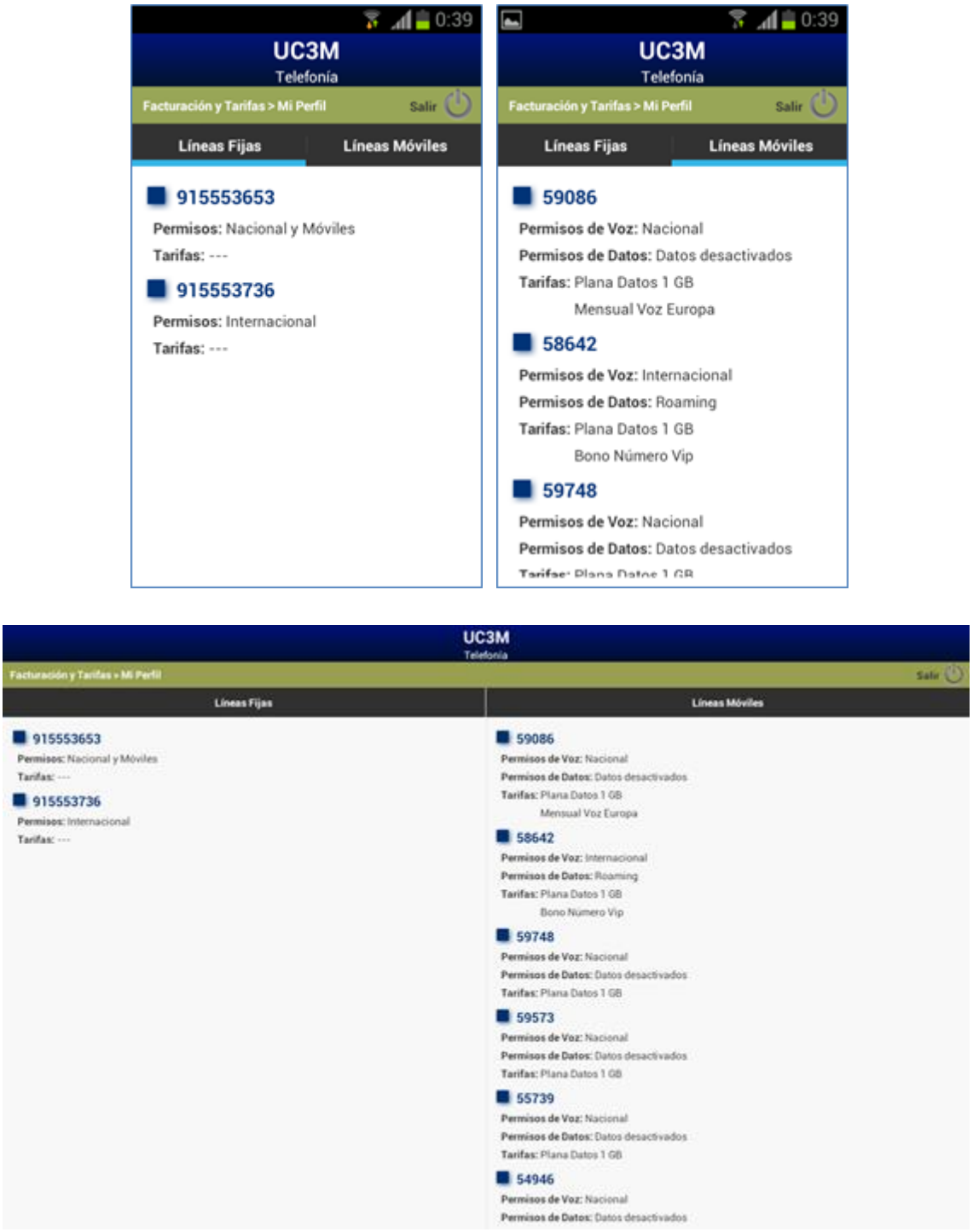

Figura 56 - Diferencia entre vista en un terminal handset (arriba) y una tablet (abajo) en Mi Perfil

Figura 57 - Diferencia entre vista en un terminal handset (arriba) y una tablet (abajo) en Consumo de Grupo

- En el caso de las guías de configuración, para que utilicen imágenes del Galaxy SII si es un handset y las de la Galaxy Tab 10.1" si es una tablet, se crean arrays (listas) en archivos XML de `TypedArray` que se guardan en sus respectivas carpetas de recursos alternativos. Estas listas contienen el identificador de los

recursos de las imágenes que se quieren utilizar. El sistema elegirá qué tipo de lista se ha de utilizar en función de en qué tipo de dispositivo se esté ejecutando (tablet o handset).

- La calidad de la pantalla y la tonalidad de los colores que se representan en ella varían de unos dispositivos a otros, por lo tanto un mismo color HTML se puede ver bastante distinto en un móvil u otro. Dado que este problema depende de cada pantalla no se puede ver reflejado en imágenes en esta memoria puesto que se verían según la tonalidad de cada pantalla donde se lea y ambas iguales.

- Para el banner que indica en qué pantalla de la aplicación se encuentra el usuario se utiliza un tamaño de letra menor que el normal de por defecto. En el caso de estar ejecutando la aplicación en un tablet, este tamaño de letra se ha de aumentar ya que es demasiado pequeño. De igual manera, el tamaño de algunos elementos gráficos han de cambiar cuando son visualizados en una tablet como el tamaño del `EditText` (campo de texto editable) para introducir el nombre y la contraseña del usuario, ya que un `EditText` que ocupase toda la pantalla es desagradable a la vista, empeorando la experiencia de uso. Para ambos casos y algunos más de esta índole, se utilizan los recursos alternativos del fichero styles.xml. Los atributos de anchura y altura, de tamaño de letra son definidos dentro de estilos que se aplican a cada vista. Además de ahorrar líneas de código en caso de que varias vistas tengan los mismos atributos ya que no habría que volverlos a escribir para cada una de ellas sino tan sólo hacer referencia al estilo en cada una de las vistas. Se pueden modificar los valores y atributos asociados a un estilo concreto para distintas opciones alternativas como cuando la aplicación se ejecuta en una tablet.

Son más comunes los problemas que acarrean las distintas versiones de Android y el querer hacer un diseño válido y similar para todas ellas, como se va a ver a continuación en alguno de los ejemplos de inconvenientes que hubo que superar durante la realización de este proyecto:

- Utilizar un archivo XML de referencias para no repetir el mismo layout en la carpeta de recursos de `values-large` (utilizado en versiones anteriores a la 3.0) y values-600swdp (utilizado en la versión 3.0 en adelante).

- Según la versión de Android, el tamaño por defecto del texto en una lista varía. En versiones anteriores a la 3.0. el tamaño del texto en las listas es similar al tamaño utilizado en el texto normal de un `TextView`. Sin embargo, desde la versión 3.0 en adelante el tamaño del texto es mayor al tamaño normal por defecto. Para tener una interfaz similar en cada uno de los dispositivos homologados en la universidad, se decidió que todos tuvieran el mismo tamaño de texto, el tamaño normal no uno mayor. Para ello en lugar de usar el layout por defecto de Android

para las listas se decidió crear un layout propio similar al que viene por defecto en Android en las versiones anteriores a la 3.0.

- Al igual que el tamaño de letra del texto visto en el anterior punto, otros elementos gráficos por defecto cambian de una versión a otra de Android. Por ejemplo algunos de los drawables incluidos en las librerías de Android cambian el diseño, tamaño y color en las diferentes versiones de Android, como el drawable `ic_lock_power_off.png`: a la izquierda en la Figura 58 se muestra su diseño para la versión 2.3.3 de Android (nivel de API 10), y a la derecha la versión 4.0.4 de Android (nivel de API 15) en el que es blanco. Ambos están con un color de fondo azul para que se vea en la memoria más fácilmente sus diferencias de diseño y color (ya que el segundo es blanco y no se vería en la memoria).

Figura 58 - Diferencia de un mismo drawable en distintas versiones de Android

Para algunos de estos cambios entre versiones de los elementos gráficos se ha decidido mantener un sólo icono metiéndolo como una imagen más en los drawables que introducimos nosotros en la aplicación en lugar de utilizar el de por defecto de Android. Este es el caso del icono de cierre de sesión visto en el ejemplo de arriba ya que afectaba de manera negativa cambiando el tamaño del banner de navegación de la aplicación. Sin embargo, otros cambios menores como el color al pulsar un botón que también difiere de una versión a otra. En este caso concreto se ha decidido mantener el color de cada uno de los dispositivos. El usuario está más acostumbrado a los colores de su propio dispositivo para estas acciones básicas y a que el esfuerzo de incluir todos los estados de un botón y todas sus imágenes repercute en una experiencia de uso peor, la aplicación se vuelve más lenta al tener que cargar más elementos y además ocupa más sin que aporte una mejoría notable o necesaria como en el caso del botón de salir de la sesión.

- El cambio del uso de colores afecta a otras funcionalidades de Android que pueden pasar más desapercibidas. El problema tiene que ver con el color primario inverso de textos a partir de la versión 3.0, que hace que el texto de error que se define en el método `setError()` de un `EditText` (campo de texto editable) no se vea en la versión 3.0 en adelante, por lo muestra la caja del mensaje de error vacía.
  En la Figura 59 se puede ver la diferencia, a la izquierda la imagen del error en las versiones anteriores a la 3.0 y a la derecha la imagen del error en versiones 3.0 en adelante:

Figura 59 - Problema con los colores primarios en el método setError de un EditText

Para solucionar esto, en el archivo XML de estilos en el que está definido el tema usado en la aplicación, se había de hacer referencia de que se utilizara para el color primario inverso del texto el color primario del tema *light* de Android. El tema base de Android del que hereda el usado en esta aplicación es el `Theme.Light.NoTitleBar`, que consiste principalmente en que el fondo de pantalla de la aplicación será blanco con letras grises oscuras y sin mostrar la barra del título de la aplicación.

- Otro problema a afrontar por los cambios en las versiones es la funcionalidad de permitir al usuario activar las redes cuando la aplicación detecta que no hay conexión a Internet. Se quería que el usuario pudiera activar tanto una red Wi-Fi como la red de datos. Para ello se crea un `Intent` en el que se indica que se abra la configuración de redes sin cables (wireless). El problema radica en que antes de la versión 3.0 tanto la activación del Wi-Fi como de las redes de paquetes de datos se hacían desde el mismo menú de configuración. Sin embargo, a partir de la versión 3.0 se sacó la activación del Wi-Fi de dicho menú y se colocó en la pantalla principal de "Ajustes" del dispositivo móvil por lo que con el intent tal y como estaba configurado sólo se podía activar las redes de paquetes ya que si se volvía hacia atrás se volvía a la aplicación de telefonía corporativa de la UC3M no a la pantalla principal de "Ajustes". Para solucionar esto, se filtró por versión de la API por código y en función del nivel de la API (versión de Android) se crea un intent que vaya al menú de configuración de redes wireless para versiones anteriores a la 3.0 o un intent que vaya al menú de ajustes a partir de la versión 3.0, ya que desde este se accede directamente al menú de configuración del Wi-Fi y se puede acceder al de datos de paquetes fácilmente también.

- Algunos componentes introducidos en versiones posteriores de Android tales como la barra de navegación, los `Fragments`, o el `ViewPager` son muy útiles y se han querido implementar en esta aplicación. Android proporciona unas librerías

de compatibilidad para que se puedan utilizar estos nuevos componentes, aunque de forma algo limitada generalmente.
En algunos casos como los de las pestañas de navegación se ha querido dar un estilo similar al estilo Holo de Android. Sin embargo, los drawables y estilos de listas y demás componentes gráficos del estilo Holo sólo están disponibles desde la versión 3.2 de Android y no están incluidos en las librerías de compatibilidad. Por lo tanto si necesita implementar algún estilo de ese tipo se habrá de recrear utilizando los medios disponibles. Por ejemplo, para implementar un botón al estilo Holo se tendrían que obtener dichas imágenes de Internet o bien extrayéndolas del propio archivo JAR con las librerías y recursos de Android de alguna de las versiones desde la 3.2 en adelante. Después, meterlas en las carpetas de recursos `drawable` alternativos según sus densidades y crear un selector de imágenes para manejar los cambios de estado del botón (presionado, normal, con el foco...) como se ve en la Figura 60:

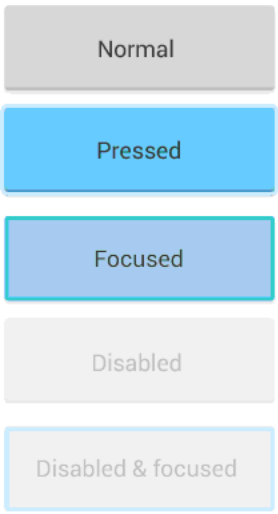

Figura 60 - Estados de un botón con el estilo Holo aplicado

Debido al aumento de tamaño de la aplicación y a la complejidad de realizar estos mismos pasos para cada uno de los componentes gráficos, se decidió mantener el estilo por defecto de cada una de las versiones de Android y dar estilo Holo sólo a ciertos componentes gráficos como las pestañas de navegación.

- Las pestañas de navegación personalizadas dieron lugar a un problema debido a las versiones algo más complejo de solucionar de forma eficaz, el problema del gráfico de división entre las pestañas. El problema radica en que si se utiliza el método `TabWidget.setDividerDrawable(int resId)` para establecer dicho gráfico, en caso de dispositivos anteriores a 3.0 si se llama al método antes de establecer las pestañas (tabs) al ejecutar la aplicación se ve el divisor y no da error al cambiar entre pestañas y si se llama al método después de establecer las pestañas da un error. El problema radica en que en ninguno de los dos casos se

ve el divisor para versiones en adelante de la 3.0 usando dicho método. Finalmente se decidió resolver utilizando un `TextView` modificado para cada pestaña. Tiene como imagen de fondo un selector de color para pintar la barra horizontal que marca qué pestaña está seleccionada, y un drawable a su izquierda que es el divisor. Además el widget que contiene todas las pestañas tiene de fondo el color gris oscuro que se requería. La primera pestaña está desplazada a la izquierda un poco para que no se vea ese primer divisor. De este modo, al no depender del método `setDividerDrawable()` el divisor es visible en todas las versiones de Android.

- Para aplicaciones 3.0 en adelante se incluyó en las librerías de Android la clase `ActionBar`. Para esa versión del sistema operativo en adelante las pestañas se asocian a la `ActionBar` de forma más directa y sencilla. En las librerías de compatibilidad no existe dicha funcionalidad por lo que, al tener que ser compatibles con versiones anteriores de Android se tuvo que utilizar el `FragmentTabHost` que implicaba mayor complejidad a la hora de codificar dicha funcionalidad.

Capítulo 4

Testing

A la hora de realizar esta aplicación se ha llevado a cabo una serie de pasos para minimizar la cantidad de posibles errores. El proceso se ha basado en crear primeramente una serie de historias de uso para cada funcionalidad. Cada historia de uso que es codificada, se prueba con los posibles casos de error y se corrige para que sea más robusta y muestre los errores pertinentes al usuario evitando que se llegue a forzar el cierre de la aplicación por un fallo. Se dio el caso de que algunas historias de uso necesitaban probarse junto con otras historias de uso por lo que se probaba el conjunto de historia de usos. Cuando una funcionalidad era codificada se probaba la funcionalidad al completo además de otras funcionalidades previamente codificadas que tuvieran cierta relación directa o indirecta para comprobar que no se hubiera roto nada de lo codificado anteriormente. Por último, se refactorizaba el código y se mejoraba allá donde se pudiera, y se volvían a pasar las pruebas para comprobar que siguiera funcionando todo correctamente tras la refactorización. Si algún bug era encontrado, era prioritario solucionarlo, o bien en el momento en el que se encontrase, o bien se anotaba como historia de uso prioritaria y se solucionaba tras completar la historia de uso con la que se estuviera en ese momento.

Además, cada día se hacía al menos un repaso general a todas las funcionalidades implementadas hasta el momento en busca de posibles errores no pensados en un primer momento y comprobar que todo siguiera funcionando correctamente.

Posteriormente se ha añadido a la aplicación un conjunto de pruebas sistemáticas utilizando las librerías de pruebas de Android, además de la librería externa Robotium.

- **Pruebas manuales en el terminal**

Las baterías de pruebas manuales en el terminal son las que se han estado realizando durante todo el desarrollo de la aplicación. Cada vez que se implementaba una nueva funcionalidad, se llevaba a cabo una refactorización o se corregía algún bug que pudiera estar relacionado directa o indirectamente con algunas de las funcionalidades ya implementadas anteriormente se comprobaba que siguiera funcionando correctamente dichas funcionalidades. Además, cada día se hacía un repaso general de todas las pruebas de las funcionalidades implementadas hasta el momento para comprobar que siguiera funcionando todo correctamente y para intentar detectar posibles errores y posibles casos que pudieran inducir a futuros errores no pensados hasta el momento.

A continuación se muestra algunas de estas baterías de pruebas recogidas en la Tabla 4.

Tabla 4 - Ejemplo de pruebas en el terminal

Nº PRUEBA	PRUEBA REALIZADA	RESULTADO	OBSERVACIONES
3.1	**Login**		
3.1.1	Entrar como PAS/PDI	OK	Se accede al menú de "Facturación y Tarifas" con un usuario y contraseña de PAS/PDI
3.1.2	Entrar como alumno da error	ERROR	No da error a la hora de acceder en el login ya que aún no se ha implementado esta limitación en el web service, pero da un error de "permisos no encontrados" si los permisos no están simulados en un archivo xml en la aplicación
3.1.3	Comprobar que se realiza la autenticación a través de una conexión segura usando Wireshark y el emulador del Eclipse	OK	Se comprueba utilizando la aplicación Wireshark y el emulador del Eclipse (para que pueda ser monitorizado el tráfico usando Wireshark) que la contraseña del usuario se manda por una conexión cifrada HTPPS
3.1.4	Cerrar sesión y vuelve a la pantalla de inicio de sesión	OK	Al pulsar sobre el botón de cerrar sesión se regresa a la pantalla de autenticación del usuario
3.1.5	Repetir 3.1.1 y al pulsar en el botón "back" vuelve a la pantalla de inicio de la aplicación	OK	Vuelve al menú inicial de la aplicación
3.1.6	Repetir 3.1.3 e iniciar sesión con otro usuario diferente y que se carguen los datos del nuevo usuario en los menús de consumo y tarifas	OK	Los datos del anterior usuario se eliminan, incluido el orden de navegación entre las pantallas de los menús al pulsar el botón "Back" del terminal, y se cargan los datos del nuevo usuario
3.1.7	Repetir 3.1.5 y que al entrar de nuevo en "Facturación y Tarifas" no pase por la pantalla de inicio de sesión ya que ya está autenticado	OK	Entra directamente al menú de "Facturación y Tarifas"
3.2	**Facturación y Tarifas**		
3.2.1	Probar datos simulados de "Mi perfil" son mostrados correctamente	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.2	Probar datos simulados de "Mi consumo" son mostrados correctamente	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.3	Probar datos simulados de "Consumo de grupo" son mostrados correctamente	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.4	Probar datos simulados de "Tarifas corporativas" son mostrados correctamente	OK	Se muestran los datos correctos acorde al archivo simulado

3.2.5	Probar datos del "Asesor de tarifas" son mostrados correctamente	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.6	Probar con permiso de grupo simulado se muestra el menú de "consumo de grupo"	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.7	Probar sin permiso de grupo simulados no se muestra el menú de "consumo de grupo"	OK	Se muestran los datos correctos acorde al archivo simulado
3.2.8	Probar todos los puntos del 3.2 con datos verdaderos obtenidos de los web service	ERROR	Aún no están implementados los web service reales

- **Pruebas sistemáticas**

Se han realizado una serie de pruebas sistemáticas, tanto pruebas unitarias para probar la funcionalidad de ciertas piezas de código de la aplicación, como test de sistema para probar la integridad del código con la interfaz gráfica.

Para las pruebas unitarias se ha hecho uso de las librerías de test de Android, basadas en JUnit 3, en concreto de la clase `InstrumentationTestCase`. No se prueba la interfaz gráfica pero sí se hace uso de la instrumentación para monitorizar la interacción entre el sistema y la aplicación. Se han hecho test para probar que cada parseador obtiene la información de los archivos XML. Se ha realizado uno por tipo de parseador existente en la aplicación, y varios archivos XML con errores para comprobar que realiza lo adecuado según el tipo de error.

Para los test de sistema, se ha querido probar la integración entre la funcionalidad del código con la representación gráfica en la interfaz y que interactúan del modo esperado. Para ello se ha hecho uso de la librería Robotium y de la clase `ActivityInstrumentationTestCase2` de las librerías de test de Android. La versión de Robotium que se ha utilizado es la última actualmente, la 4.3.

Un ejemplo de este tipo de test es en el que se ha probado las pestañas de la pantalla de “Guía de buenas prácticas” de la aplicación. Dado que esta pantalla tiene pestañas para la navegación y hace uso de fragments para mostrar el contenido se han diseñado los test para que prueben que al seleccionar en cada pestaña se carga el fragment adecuado, que éste es visible, y además que muestra la información que el cliente quiere que aparezca. Además se prueba que el texto del banner es el correcto.

Estas pruebas podrían hacerse directamente con las clases derivadas de `Instrumentation` y otras clases del paquete de test de Android, pero la librería Robotium facilita la elaboración de los test. Además, Robotium permite test de caja negra, por lo que no se ha de conocer cómo está implementado el código en detalle, si no que se

puede utilizar directamente instrucciones tales como “presiona el texto que ponga Ahorro de voz” (`solo.clickOnText("Ahorro voz"))`, que facilitan en gran medida el pulsar sobre una de las pestañas del ejemplo expuesto.

Como estos test se prueban interactuando sobre la interfaz gráfica directamente (de hecho se puede ver los pasos en el dispositivo móvil mientras las pruebas se van ejecutando) son mucho más lentos que las pruebas unitarias con JUnit 3. Sobre todo la acción de buscar un texto en la pantalla visible es muy costosa en tiempo.

Para las pruebas se creó un proyecto de test en eclipse separado del proyecto principal de la aplicación de telefonía corporativa de la UC3M, pero dependiente de éste, por lo que puede hacer uso de las clases y métodos del proyecto principal para probarlos. Además, de este modo el usuario final sólo ha de instalar el .apk del proyecto principal, ocupando un menor espacio la aplicación en el dispositivo móvil del usuario. Y por otro lado, permite añadir por ejemplo distintos archivos XML con errores para los test de los parseadores, sin mezclar con los archivos XML correctos.

Cualquier clase de test en Android ha de implementar los métodos `setUp()`, donde se prepara lo necesario para el test, como la inicialización de variables; y el método `tearDown()`, donde se cerrarán todos los posibles elementos que sean necesarios cerrar, como bases de datos o actividades. Cada uno de los test, serán métodos públicos que han de comenzar con la palabra test, como por ejemplo `testBanner()`. Los métodos que no comiencen con la palabra “test” no serán considerados test si no métodos auxiliares para los test.

Para crear los tests de sistema, se ha de crear un objeto `Solo`, perteneciente a la librería de Robotium pasándole como argumentos el objeto `Instrumentation` y la `Activity` que se quiere probar. Todos los métodos relacionados con la interfaz gráfica y la instrumentación se realizan llamándose sobre el objeto de la clase Solo, como por ejemplo `Solo.asserCurrentActivity(String message, Class activityClass)` o `Solo.clickOnText(String text)`.

Además, si se está probando una actividad como en el caso de los test de sistema creados, es imprescindible el incluir el constructor de la clase de test con una llamada al método `super` pasándole como argumento el nombre de la clase `Activity` a probar, como por ejemplo `super(DeviceConfiguration.class)`.

Capítulo 5

Historia del proyecto

5.1. Tiempos

Este proyecto se ha realizado durante el periodo de trabajo de la autora del proyecto en el área de telefonía de la Universidad Carlos III de Madrid, lo cual ha marcado significativamente los tiempos de realización de dicho proyecto.

Este proyecto se ha realizado en un tiempo aproximado a un año, desde el inicio de su concepción a través de los primeros mockups hasta la finalización de la presente memoria. Sin embargo, todo este tiempo no se corresponde con una dedicación por completo a este proyecto debido a varios motivos. El primero de ellos es que el proyecto se ha realizado mayormente durante las horas de trabajo (30 horas semanales) excepto la presente memoria que se ha realizado posteriormente y en las últimas semanas en las que se dedicó tiempo fuera de las horas de trabajo. Además, durante las horas de trabajo se debía realizar muchas otras tareas no relacionadas con este proyecto, por lo que tampoco se ha trabajado en dicho proyecto esas 30 horas semanales enteramente. Por otro lado, aunque el comienzo del proyecto fue a primeros de octubre del 2012 cuando se inició la creación de los primeros mockups, esos primeros meses fueron más de espera hasta que los asesores de contenido de la UC3M validaran dichos mockups, por lo que la implementación del código y el comienzo efectivo del proyecto no empezó hasta dos meses después, a primeros de diciembre del 2012.

En la Figura 61 y la Tabla 5 se puede observar el diagrama de Gantt con las fases del proyecto y el tiempo empleado en cada una. En las siguientes subsecciones se explican brevemente cada una de estas fases.

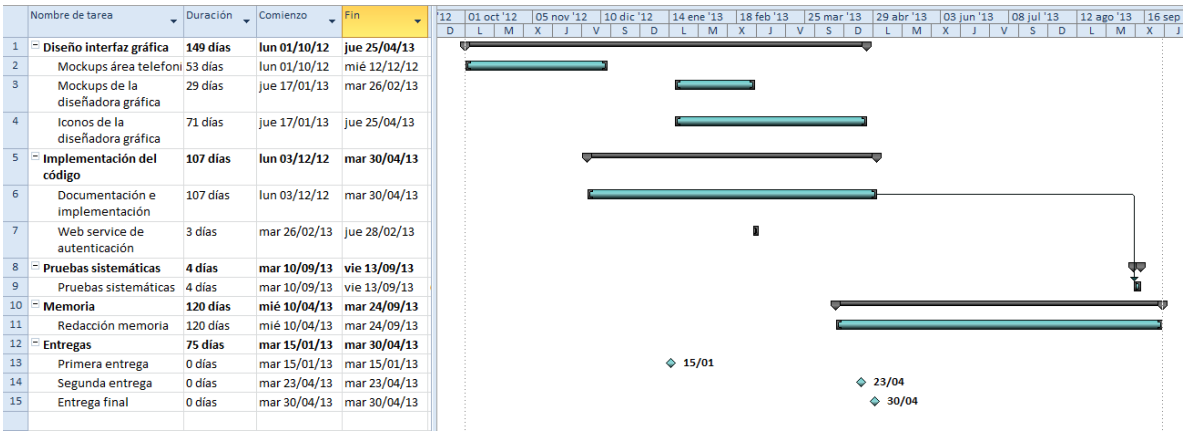

Figura 61 - Diagrama de Gantt del proyecto

Tabla 5 - Fases del desarrollo en el diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Diseño interfaz gráfica**	**149 días**	**lun 01/10/12**	**jue 25/04/13**	
2	Mockups área telefoni	53 días	lun 01/10/12	mié 12/12/12	
3	Mockups de la diseñadora gráfica	29 días	jue 17/01/13	mar 26/02/13	
4	Iconos de la diseñadora gráfica	71 días	jue 17/01/13	jue 25/04/13	
5	**Implementación del código**	**107 días**	**lun 03/12/12**	**mar 30/04/13**	
6	Documentación e implementación	107 días	lun 03/12/12	mar 30/04/13	
7	Web service de autenticación	3 días	mar 26/02/13	jue 28/02/13	
8	**Pruebas sistemáticas**	**4 días**	**mar 10/09/13**	**vie 13/09/13**	
9	Pruebas sistemáticas	4 días	mar 10/09/13	vie 13/09/13	6
10	**Memoria**	**120 días**	**mié 10/04/13**	**mar 24/09/13**	
11	Redacción memoria	120 días	mié 10/04/13	mar 24/09/13	
12	**Entregas**	**75 días**	**mar 15/01/13**	**mar 30/04/13**	
13	Primera entrega	0 días	mar 15/01/13	mar 15/01/13	
14	Segunda entrega	0 días	mar 23/04/13	mar 23/04/13	
15	Entrega final	0 días	mar 30/04/13	mar 30/04/13	

- **Mockups del área de telefonía (2 meses)**

En esta fase del proyecto se realizaron las primeras versiones de los mockup con lo que en el departamento se deseaba que incluyese la aplicación. Cada versión era revisada por los asesores de contenidos de la UC3M y dictaban pautas y directrices de diseño para la siguiente versión.

- **Mockups de la diseñadora gráfica (1 mes)**

En esta fase, se encargó la diseñadora gráfica de realizar las últimas versiones de los mockups. Aunque algunas pantallas y requisitos de información a mostrar variaron posteriormente a la última versión de los mockups no se creó una nueva versión actualizada, por lo que se mantiene en alrededor de un mes el tiempo de ejecución de esta fase.

- **Iconos de la diseñadora gráfica (3 meses)**

En esta fase, la diseñadora gráfica de la UC3M realizó la iconografía necesaria para la aplicación, conjuntamente con la creación de los mockups.

- **Documentación e implementación del código (5 meses)**

Dado que ya se tenía un conocimiento previo de Android no se precisó de una documentación exhaustiva antes de comenzar la implementación del código. Sin embargo ha sido necesario documentarse a menudo para ciertas necesidades, sobre todo con relación a las nuevas librerías de compatibilidad de Android en el uso de fragments para el diseño dinámico en smartphones y tablets. También para la creación de conexiones seguras para el envío de contraseñas, el parseado de archivos xml o para crear el efecto de pinch to zoom en las imágenes de las guías de configuración. Por lo tanto es un elemento estrechamente unido a la propia implementación del código y difícilmente calculable.

La implementación y documentación se realizó paralelamente al diseño e iconografía de la aplicación finales por parte de la diseñadora gráfica, para que no supusiera un retraso en la implementación. Posteriormente fue adaptado a dichos mockups o lo más posible dentro de la funcionalidad necesaria.

- **Entregas y reuniones**

Como en todo proyecto real era necesario la entrega de versiones lo más estables y funcionales posibles en ciertas fechas para la reunión con los clientes. Para esta aplicación el rol de cliente lo han ejercido los superiores encargados del proyecto por parte del área de telefonía y de los asesores de contenido de la UC3M. En las reuniones se validaba y/o proponían cambios en el diseño, contenido y funcionalidad de la aplicación.

- **Web service de autenticación (2 días)**

Creación del web service de autenticación y pruebas de su funcionamiento.

- **Pruebas sistemáticas (4 días)**

Durante todo el proceso de la implementación del código se llevan a cabo pruebas funcionales manualmente sobre la aplicación para testear que todo lo existente funcione correctamente después de un nuevo cambio en la aplicación. Sin embargo, esta fase de pruebas se refiere al periodo de creación de las pruebas sistemáticas de la aplicación, que supusieron un añadido posterior al proyecto, fuera ya del trabajo realizado durante el periodo de trabajo en la universidad.

- **Redacción de la memoria (5 meses)**

Al igual que durante el desarrollo de la aplicación, el tiempo real dedicado a la redacción de la memoria es menor, concentrándose prácticamente su totalidad en los meses de mayo, junio, julio y septiembre

5.2. Presupuesto

En este apartado se presenta el presupuesto necesario para este proyecto. Se puede desglosar en costes de personal y coste de material.

- **Costes de personal**

Los costes de personal incluyen el salario medio de la Ingeniera Técnica de Telecomunicaciones: Sonido e Imagen encargada del desarrollo del proyecto. Se ha considerado para este cálculo el salario medio de un programador según la web Infojobs Trends [23], pero al tratarse de un programador junior el sueldo medio es algo menor por lo que se ha decidido dejar en 15.000 € al año, en 12 pagas por lo que dejaría un coste de alrededor de 1250 € al mes. Debido a la crisis actual y a la disminución de sueldos en general, el salario medio real será seguramente aún menor, por lo que los redondeos de los cálculos se harán haciendo hacia abajo.

Dado que la autora de este proyecto tan sólo trabajaba durante 6 horas durante su trabajo en el área de telefonía de la universidad, y el cálculo anterior del salario medio es sobre

un trabajo de 8 horas diarias y dado que el tiempo empleado en el desarrollo del proyecto durante esos días de trabajo y en los meses posteriores no ha sido a tiempo completo, se ha decidido estimar en 132 días reales de trabajo (8 horas) para hacer el cálculo de los honorarios, es decir, alrededor de 6 meses de salario.

Se ha decido estimar el tiempo de trabajo de la diseñadora gráfica al no poder conocer el tiempo real de horas dedicadas al diseño de este proyecto. Se ha considerado de un mes el tiempo de trabajo. Para el cálculo del salario se ha considerado el salario medio de un diseñador web según la web de Infojobs Trends [24]. Este valor asciende alrededor de 13.700 € al año, que dividido en 12 pagas, tendría un coste mensual de unos 1100 €.

Para el coste del salario del desarrollador de aplicaciones web encargado del web service de autenticación se ha decidido considerar el coste por hora de un programador sénior.[25] valorado en 29 €. Se ha decidido estimar que el tiempo de trabajo fue de 5 horas.

En la Tabla 6 se puede ver reflejados estos datos.

Tabla 6 - Costes de personal

Concepto	Meses	Honorarios	Importe
Desarrollador de software Jr.	6	1250 €/mes	7500 €
Diseñadora gráfica	1	1100 €/mes	1100 €
Programador sénior	5 horas	29 €/hora	145 €
Total			**8745 €**

- **Costes de material**

En esta sección se desglosa el coste del material utilizado durante el desarrollo de este proyecto.

Ordenador proporcionado en el área de telefonía de la UC3M, valorado en unos 350 € [26].

- Smartphone con versión 4.0 en adelante de Android: Samsung Galaxy SII, valorado en 308,39€ [27].

- Smartphone con versión anterior a la 4.0 de Android: HTC Desire HD, valorado en 349€ [28].

Tablet: Galaxy Tab 10.1, valorada en 425€ [29].

- Conexión a internet, necesaria tanto para la búsqueda de documentación, como para probar la aplicación haciendo uso del Wi-Fi. Se ha valorado sobre la tarifa de un usuario particular como es la autora de este proyecto y no sobre el coste que tiene actualmente la universidad al no poder tener un dato aproximado real de dicho coste. Este valor asciende a 40 € mensuales [30], que multiplicado por el número de meses de trabajo (se considera que los dos primeros meses de mockups son despreciables el tiempo real de trabajo utilizando internet) supone un coste de 400 €.

En la Tabla 7 se recogen todos estos datos para hacer un resumen de los costes de material:

Tabla 7 - Costes de material

Concepto	Precio/unidad	Importe
Ordenador	350 €	350 €
Samsung Galaxy SII	308,39 €	308,39 €
HTC Desire HD	349 €	349 €
Samsung Galaxy Tab 10.1	425 €	425 €
Conexión internet	40 €/mes	240 €
Total		**1672,39 €**

- **Presupuesto total**

El presupuesto total está compuesto por el coste de personal y el coste de material, ascendiendo a un total de 6832,39 €, tal y como se detalla en la Tabla 8:

Tabla 8 - Presupuesto total

Concepto	Importe
Costes de personal	8745 €
Costes de material	1672,39 €
Total	**10.417,39 €**

Capítulo 6
Conclusiones

En este proyecto se ha realizado la futura aplicación de telefonía de la Universidad Carlos III de Madrid para el sistema operativo móvil Android, acercando algunos de los servicios del área de telefonía de la universidad a sus empleados, de forma que resulten más cómodos, sencillos y accesibles desde cualquier lugar. Se ha trabajado para conseguir un diseño multidispositivo adecuado y una buena usabilidad.

En el momento de la redacción de esta memoria aún no se ha puesto a disposición de los empleados de la universidad la aplicación, ya que la empresa encargada de los web services del área de "Facturación y Tarifas" aún no los ha creado. Pero espero que pueda ser de utilidad una vez entre en funcionamiento.

Tras concluir el proyecto de telefonía corporativa de la UC3M queda hacer un repaso de lo que ha sido este proyecto, los objetivos cumplidos, así como las líneas de trabajo futuras.

Me siento satisfecha con los objetivos cumplidos en este proyecto, a pesar de que no haya podido ser finalizada por completo la aplicación al depender de una empresa externa los web services del área de "Facturación y Tarifas", la que considero la funcionalidad más importante de la aplicación, ya que será la más utilizada. Al menos ya está preparada la aplicación para la conexión con los web service reales. Sin embargo, me hubiera gustado poder haberla visto completada y en uso por los trabajadores de la universidad. Hubiera sido muy provechoso obtener el *feedback* de dichos usuarios para saber en qué mejorar la experiencia de uso.

A continuación se habla sobre qué realicé o di soporte técnico y/o de diseño en el proyecto, sobre mis objetivos personales cumplidos, sobre qué haría de otro modo si hubiera de realizar de nuevo este proyecto y por último de las líneas futuras a seguir.

6.1. ¿Qué he hecho?

Para el desarrollo de este proyecto, debido a su carácter de aplicación oficial de la UC3M, se ha necesitado colaborar con diversas personas en su creación. En esta sección se aclara en qué partes del desarrollo he estado involucrada y en cuáles no.

Principalmente me encargué de toda la programación de la aplicación en Android y la creación de la batería de pruebas manuales y las pruebas sistemáticas. También ayudé a elaborar los primeros mockups y tomar decisiones de diseño en diversas pantallas que no habían sido tenidas en cuenta en los mockups o en otras que tuvieron que ser adaptadas por la mala usabilidad del diseño original.
No he sido partícipe en la creación del web service de autenticación. Por lo contrario, sí que he creado la base para crear los web service del área de "Facturación y Tarifas" al diseñar los archivos XML de ejemplo que se usan en la aplicación actualmente, en los cuales habrá de basarse la empresa externa encargada de su implementación real.

A la hora de la codificación de la aplicación, he hecho uso de código ya implementado, siguiendo la idea de reutilización de código inherente a cualquier API, pero adaptándolo al uso y necesidades de esta aplicación. Es el caso de la funcionalidad de *pinch to zoom* por ejemplo, que no ha sido codificada desde cero. De igual modo, otras funcionalidades como el cliente de HTTPS o las pestañas usando el `FragmentTabHost` han sido fruto de la búsqueda en diversos sitios y comunidades de Internet, especialmente en la web stackoverflow[11], y adaptando, mezclando y mejorando las ideas propuestas y los ejemplos encontrados a esta aplicación.

6.2. ¿Qué he aprendido?

Durante el desarrollo de la aplicación de telefonía corporativa de la UC3M he podido afianzar y mejorar mis conocimientos de Android previos. Por ejemplo, hacer un uso más inteligente de las carpetas de recursos alternativos o cómo manejar adecuadamente la pila de tareas y actividades. Sobre todo me ha permitido empezar a conocer los últimos elementos introducidos en el API de Android, tan útiles como `Fragments` o `ViewPager`, que permiten una interfaz y experiencia de uso mucho más rica y moderna.

Además, me ha ayudado a aprender a resolver la problemática de lidiar con la fragmentación de Android, tanto según el tipo de dispositivos móviles como de versiones de Android y a trabajar utilizando las librerías de compatibilidad de Android.

También a afianzar mi conocimiento sobre parseadores de respuestas de web service de terceros para la comunicación de información, técnica útil y muy aplicada actualmente.

Más que aprender, han sido reafirmar una vez más que el uso de sistemas de control de versiones son una herramienta bastante esencial, tanto en la corrección de errores, como en la recuperación de piezas de código que se pensaron que ya no se necesitarían más.

[11] Web stackoverflow: http://stackoverflow.com/

También para poder trabajar de forma cómoda desde varios ordenadores y tener siempre una copia de seguridad no dependiente de un solo ordenador.

Por otro lado, me ha permitido aprender a realizar una aplicación para un uso real desde cero, y trabajar conjuntamente junto a otros profesionales para su consecución satisfactoria.

Estoy muy satisfecha con el diseño de las pantallas que tuve que idear dada simplemente la información que se quería mostrar tales como el “Asesor de Tarifa” o la “Normativa de Telefonía”. Ha quedado la interfaz limpia y bonita que buscaba aún en casos de tener que mostrar mucha información en la pantalla.
El utilizar una serie de buenas prácticas, tales y como los comentarios, nombres de variables y métodos descriptivos y las refactorizaciones, ha permitido obtener código más claro, organizado y limpio, que a la hora de revisar ciertas funcionalidades pasado un tiempo ha permitido llevar a cabo estas tareas de forma mucho más rápida. Además, ha facilitado en gran medida el paso de la aplicación al desarrollador encargado de mantener y continuar las líneas futuras necesarias para el correcto funcionamiento de la aplicación (como conectar con los web service reales una vez estén hechos), no siendo necesaria mucha explicación del código de la aplicación en el proceso.

Por último, he podido comprobar la utilidad de las pruebas sistemáticas al poder realizar las mismas pruebas cambiando condiciones del escenario de forma rápida sin necesidad de tener que cambiarlas manualmente en el código o en el dispositivo móvil. Pues se trata de una tarea pesada y que puede provocar errores en caso de olvidarse de dejar todo igual que al principio de las pruebas. También he podido aprender a realizar pruebas de sistema con Robotium, que en verdad hace que sea muy fácil y rápido y permite realizar test de caja negra fácilmente.

6.3. ¿Qué haría de otra forma si tuviera que empezar de cero?

Una vez finalizado un proyecto de cualquier tipo siempre hay cosas que, gracias al conocimiento adquirido durante el desarrollo del proyecto, haríamos de otro modo, y este proyecto no es una excepción. A continuación se enumeran algunas de las cosas que realizaría de otro modo si tuviera que empezar de nuevo con el mismo proyecto:

- Utilizar `Fragments` desde un principio en lugar de `Activity` simple en algunas clases como la de detalles de “Mi Consumo” o en los `ViewPager` de las guías de configuración.

- Implementar excepciones propias para el control de errores en la propia aplicación que explicasen el fallo para un mayor manejo del control de errores. Sobre todo en los parseadores, en los que ahora mismo sólo se devuelve el objeto correcto o null pero no un error en concreto fácil de manejar las consecuencias y mostrar mensajes más completos al usuario.

- Utilizar varias clases de `AsyncTask` en lugar de sólo tener una para obtener los datos de internet y hacer el resto de procesos que han de ir en otro hilo a través de instancias de la clase `Thread`.

- No perder tanto tiempo investigando cómo realizar la funcionalidad de *Pinch to Zoom* en las imágenes, directamente haber usado un `WebView` que es lo que se suele utilizar aunque no esté hecha dicha clase para esta finalidad concretamente.

- Implementar mucho antes el autocompletado, ya que era sencillo y no llevaba mucho tiempo, y hubiera supuesto un gran ahorro de tiempo a la hora de la realización de pruebas de la aplicación en el dispositivo móvil.

- Aprender a usar Git que es más rápido que SVN y en cualquier caso, me hubiera venido bien aprender a utilizarlo pues se utiliza en muchísimos proyecto hoy en día.

- Hacer la memoria a la vez que avanza el proyecto y no al final, pues se hace pesado.

6.4. Líneas futuras

Como conclusión a esta memoria se va a hablar sobre las líneas futuras que ha de seguir la aplicación y algunas posibles mejoras futuras.

El principal cambio a realizar es conectar la aplicación con los web services del área de “Facturación y Tarifas” una vez que estos hayan sido creados por la empresa externa a la universidad a la que han sido encargados. La aplicación está preparada para este cambio ya que todo el código necesario (excepto la URL del webservice) ya está implementado pero comentado. El código que por otra parte ha de ser borrado o comentado para que los datos no se sigan obteniendo del fichero local de la carpeta asset del proyecto está claramente acotado entre comentarios para facilitar esta conexión con los web services.

Otro cambio importante no implementado aún, es cerrar el acceso de inicio de sesión a los alumnos. Esto debe hacerse en el web service de autenticación para que sólo realice la búsqueda de las credenciales en el LDAP excluyendo la rama de los estudiantes. Este

cambio está pendiente de que lo realice el departamento de la UC3M encargado de dicho web service. Una vez se conecte la aplicación con los web service reales de facturación y tarifas, en el caso de que un alumno inicie sesión (si no se ha cerrado aún el servicio) simplemente dará un error de que no ha encontrado datos de ese usuario tal y como sucede ahora mismo con los archivos XML emulados.

Por otro lado, como trabajo futuro de mantenimiento de la aplicación se ha de actualizar el archivo XML con el texto de los avisos de telefonía cuando sea necesario. Además se ha de actualizar los posibles cambios de tarifas en el asesor de tarifas.

A continuación se van a enumerar otras posibles mejoras que no condicionan la correcta funcionalidad de la aplicación, pero pueden mejorarla sustancialmente:

- Mejorar el diseño para tablets, adaptando aquellas pantallas que no están del todo adaptadas y mejorando el diseño de las ya existentes.

- Utilizar técnica de single sign on, en toda la aplicación para que no sea necesario volver a iniciar sesión en otras zonas de la aplicación que abren una página web de la universidad en la que el usuario se ha de validar de nuevo, como en el caso de querer activar o modificar la cuenta de VoIP.

- Otro cambio apropiado, sería el disminuir el tamaño de la aplicación. El tamaño de la aplicación depende en mayor medida del tamaño de las imágenes de las guías de configuración. Ahora mismo la aplicación almacena todas las imágenes, tanto las de dispositivos tablets como handset, pero sólo muestra aquellas válidas para el dispositivo en el que se ejecuta. La mejora consistiría en tener un tamaño reducido de la aplicación sin imágenes, realizar una comprobación del tipo de terminal en el que se está ejecutando, y descargar las imágenes adecuadas al tamaño del dispositivo. Este proceso sólo se llevaría a cabo la primera vez que se ejecutase la aplicación.

- Dar la posibilidad de ver la guía de configuración que se desee no sólo la del dispositivo en el que se esté ejecutando.

- Realizar más pruebas sistemáticas y aprender a utilizar los objetos Mock (objetos simulados) en pruebas unitarias.

- Crear excepciones propias, sobre todo en las clases de los parseadores, para poder realizar un mejor manejo de los posibles errores de la aplicación y dar información más precisa al usuario sobre el error.

- Traducir la aplicación al inglés.

Apéndice

Glosario de Términos

API	Application Programming Interface
ADT	Android Developer Tools
CASO	Centro de Atención y Soporte de la Universidad Carlos III de Madrid
GIF	Graphics Interchange Format
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Enviroment
IP	Internet Protocol
JPEG	Joint Photographic Experts Group
LDAP	Lightweight Directory Access Protocol
PNG	Portable Network Graphics
SCV	Sistema de control de versiones
SVN	Subversion
UC3M	Universidad Carlos III de Madrid
URI	Uniform Resource Identifier
VoIP	Voice over IP
VPN	Virtual Private Network
XML	eXtensible Markup Language

Bibliografía

[1] Ventas de smartphones frente a móviles normales en el primer trimestre del 2013. http://www.xatakamovil.com/mercado/idc-samsung-domina-el-primer-trimestre-por-primera-vez-se-venden-mas-smartphones

[2] Tabla de ventas de smartphones de los años 2012 y 2013 y el porcentaje de mercado que abarca los cinco principales sistemas operativos de móviles. http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[3] Término “Android” en Wikipedia: http://en.wikipedia.org/wiki/Android_(operating_system)

[4] Dibujo de la evolución en Android: http://www.xatakamovil.com/sistemas-operativos/un-dibujo-sobre-la-evolucion-de-android-de-un-empleado-de-google-incluye-a-key-lime-pie

[5] Distribución de las versiones de Android en el mercado. http://developer.android.com/about/dashboards/index.html

[6] Arquitectura de Android: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android

[7] Replicant: http://en.wikipedia.org/wiki/Replicant_(operating_system)

[8] SQL: http://es.wikipedia.org/wiki/SQL

[9] Cualificadores de los directorios de los recursos alternativos en Android (tabla 2) http://developer.android.com/guide/topics/resources/providing-resources.html

[10] Colores en formato hexadecimal: http://www.w3schools.com/tags/ref_colorpicker.asp

[11] Diagrama del ciclo de vida de una Activity: http://developer.android.com/reference/android/app/Activity.html

[12] Android developers, pila de actividades y tareas: http://developer.android.com/guide/components/tasks-and-back-stack.html

[13] Diagrama de cómo funciona el mecanismo para salvar el estado de una Activity: http://developer.android.com/intl/es/guide/components/activities.html

[14] Fragments:
http://developer.android.com/guide/components/fragments.html

[15] Interfaz de usuario en Android:
http://developer.android.com/guide/topics/ui/overview.html

[16] Código ISO 639-1 para códigos del lenguaje en recursos alternativos (segunda columna de la tabla)
http://www.loc.gov/standards/iso639-2/php/code_list.php

[17] Código ISO 3166-1-alpha-2 para códigos de las regiones del lenguaje en recursos alternativos http://www.iso.org/iso/prods-services/iso3166ma/02iso-3166-code-lists/country_names_and_code_elements

[18] Robotium: https://code.google.com/p/robotium/

[19] Assembla: https://www.assembla.com/home

[20] API y tutoriales de Robotium: https://code.google.com/p/robotium/w/list

[21] Patrón de diseño Data Drill Down:
http://www.androidpatterns.com/uap_pattern/data-drill-down

[22] XML Parser en Android Developers:
http://developer.android.com/training/basics/network-ops/xml.html

[23] Salario medio programador en Infojobs Trends:
http://plandecarrera.infojobs.net/puesto-de-trabajo/programador

[24] Salario medio diseñador gráfico en Infojobs Trends
http://plandecarrera.infojobs.net/puesto-de-trabajo/disenador-grafico

[25] Coste por hora de un programador sénior:
http://www.boe.es/boe/dias/2007/10/11/pdfs/B11987-11987.pdf

[26] Datos proporcionados por técnicos del CAU (centro de atención al usuario) de la UC3M

[27] Precio del Samsung Galaxy S II en Amazon.es:
http://www.amazon.es/Samsung-Galaxy-I9100-Smartphone-procesadores/dp/B004Q3QSWQ

[28] Precio del HTC Desire HD en Amazon.es:

http://www.amazon.es/HTC-Desire-HD-Tel%C3%A9fono-M%C3%B3vil/dp/B004A1TYAM/ref=sr_1_26?ie=UTF8&qid=1378178814&sr=8-26&keywords=htc+desire+hd

[29] Precio de la Galaxy Tab 10.1 en Amazon.es:
http://www.amazon.es/Samsung-Galaxy-Tab-10-1-Pixeles/dp/B006L5P1DE/ref=sr_1_38?s=computers&ie=UTF8&qid=1378179058&sr=1-38&keywords=galaxy+tab+10

[30] Coste mensual de la tarifa de 20 Mb de Ono, tras la oferta inicial de un año, y sumándole el coste mensual de la línea no incluidas en la oferta de la web (ver condiciones del combinado)
https://www.ono.es/tienda/internet-telefono/?prd=1402